
OMERO

The Open Microscopy Environment

Feb 28, 2024

CONTENTS

1	OMERO Overview and CLI User Documentation	3
2	System Administrator Documentation	77
3	Developer Documentation	353
	Index	763

The documentation for OMERO 5.6.10 is divided into three parts:

OMERO Overview and CLI User Documentation introduces the user-facing client applications and how to get started, details the CLI client, and indicates where users can access further help and support.

System Administrator Documentation includes instructions for installing and configuring an OMERO server and also information on managing users and data, a task which full system administrators can now delegate to facility managers or other trusted users using the new ‘restricted administrator’ role.

Developers can find more specific and technical information about OMERO in the *Developer Documentation*.

Additional online resources can be found at:

- [Downloads](#)
- [Security Advisories](#)
- [User guides](#)
- [OME YouTube channel](#) for tutorials and presentations
- [Demo server](#) - managed by the main OME team, providing the latest released versions of OMERO and plugins for you to try out
- OMERO API documentation - [OmeroJava API](#), [OmeroPy API](#), [OmeroBlitz / Slice API](#)

OMERO 5.6.10 uses the [June 2016 schema](#) of the [OME Data Model](#). The [CHANGELOGS](#) page details the development of OMERO functionality over time.

A summary of breaking changes and new features for 5.6.10 can be found on the pages below:

- [What’s new for OMERO users](#)
- [What’s new for OMERO sysadmins](#)
- [What’s new for OMERO developers](#)

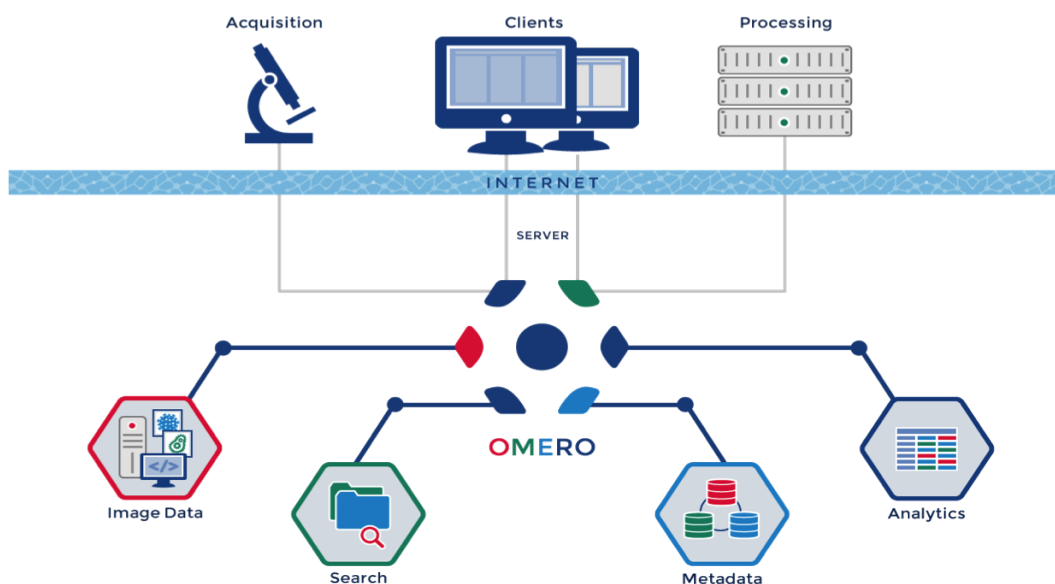
The source code is hosted on Github. To propose changes and fix errors, go to the [documentation repository](#), fork it, edit the file contents and propose your file changes to the OME team using [Pull Requests](#). Alternatively, click on “Edit on GitHub” in the menu.

OMERO OVERVIEW AND CLI USER DOCUMENTATION

1.1 Introduction

OME Remote Objects (OMERO) is a modern client-server software platform for visualizing, managing, and annotating scientific image data. OMERO lets you import and archive your images, annotate and tag them, record your experimental protocols, and export images in a number of formats. It also allows you to collaborate with colleagues anywhere in the world by creating user groups with different permission levels. OMERO consists of a Java server, several Java client applications, as well as Python and C++ bindings and a Django-based web application.

The OMERO clients are cross-platform. To run on your computer they require Java 8 or higher to be installed. This can easily be installed from <https://java.com/> if it is not already included in your OS. The OMERO.insight client gets all of its information from a remote OMERO.server — the location of which is specified at login. Since this connection utilises a standard network connection, the client can be run anytime the user is connected to the internet.



This documentation is for the OMERO 5 Platform. This version is designed to improve our handling of complex multidimensional datasets. It allows you to upload your files in their original format, preserving file names and any nested directory structure in the server repository. For more technical information, please refer to the *Developer Documentation*. You can read about the development of OMERO in the *CHANGELOGS* and the latest user-facing changes in *What's new for OMERO 5.6 for users*.

1.2 OMERO clients

1.2.1 OMERO clients overview

Most laboratories use a number of different imaging platforms and thus require tools to manage, visualize and analyze heterogeneous sets of image data recorded in a range of file formats. Ideally a single set of applications, running on a user's laptop or workstation, could access all sets of data, and provide easy-to-use access to this data.

OMERO ships as a server application called **OMERO.server** and a series of client applications (known simply as clients): **OMERO.web**, **OMERO.insight** and **OMERO.importer**. All run on the major operating systems and provide image visualization, management, and annotation to users from remote locations. With a large number of OMERO.server installations worldwide, OMERO has been shown to be relatively easy to install and get running.

OMERO.insight and OMERO.importer are desktop applications written in Java and require Java 8 (or higher) to be installed on the user's computer (this can easily be installed from <https://java.com/> if it is not already included in your OS).

Our [User guides](#) provide a series of workflow-based guides to performing common actions in the client applications, such as importing and viewing data, exporting images, drawing and analyzing of ROIs.

Our partners within the OME consortium are also producing new clients and modules for OMERO, integrating additional functionality, particularly for more complex image analysis. See the [features pages](#) for more details.

Features

Among many features, the noteworthy elements of the two main clients (OMERO.insight and OMERO.web) are:

- DataManager, a traditional tree-based view of the data hierarchies in an OMERO.server. DataManager supports access to all image metadata, annotations, tags etc.
- ImageViewer, for visualization of 5D images (space, channel, time). The ImageViewer makes use of the OMERO.server's Rendering Engine, and provides high-performance viewing of multi-dimensional images on standard workstations (e.g. scrolling through space and time), without requiring installation of high-powered graphics cards. Most importantly, image viewing at remote locations is enabled. Image rendering settings are saved and chosen by user ID
- Working Area, for viewing, annotating, and manipulating large sets of image data
- user and group administration

OMERO.web

OMERO.web is a web-based client for users who wish to access their data in the browser. This offers a similar view to the OMERO.insight desktop client. Figures [OMERO.web user interface](#) and [OMERO.web image viewer](#) present the user interface. Developers can use the [OMERO.web framework](#) to build customized views.

OMERO.web features almost all of the functionality of OMERO.insight barring import. A number of apps are available to add functionality to OMERO.web, such as [OMERO.figure](#) and [OMERO.iviewer](#). See the main website for a [list of released apps](#).

For more information and guides to using OMERO.web, visit [User guides](#).

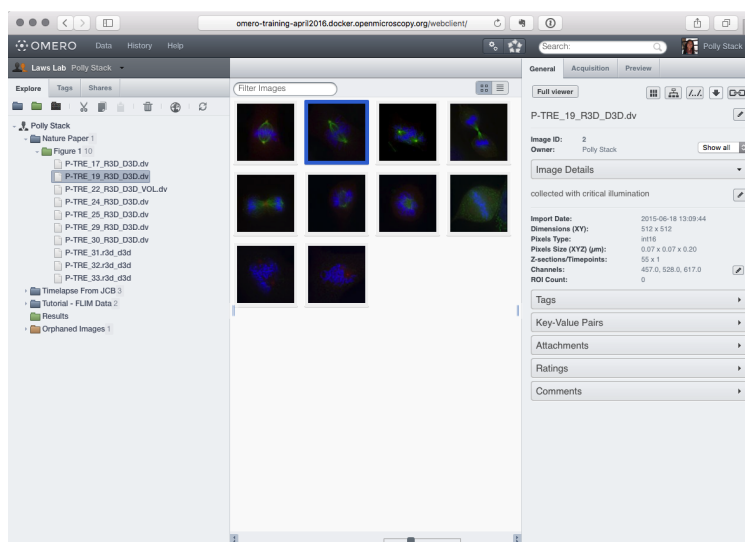


Fig. 1: OMERO.web user interface

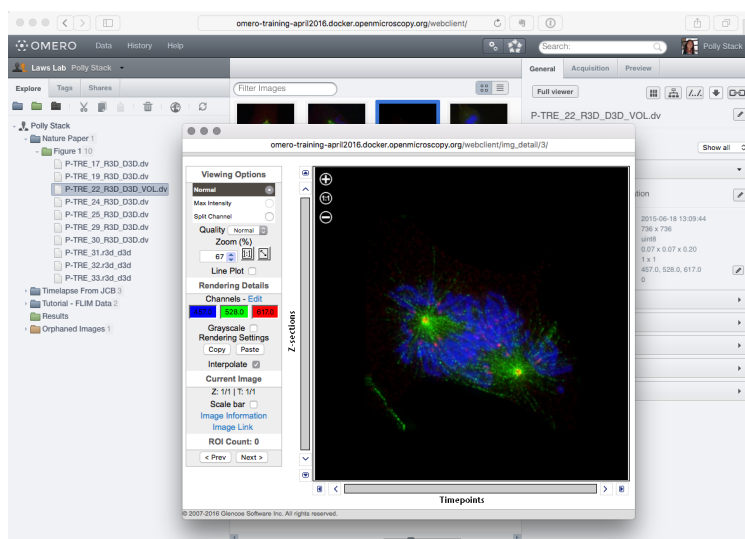


Fig. 2: OMERO.web image viewer

OMERO.insight

Note: With the release of OMERO 5.3.0, the OMERO.insight desktop client has entered **maintenance mode**, meaning it will only be updated if a major bug is discovered. Instead, the OME team will be focusing on developing and extending the web clients.

OMERO.insight provides a number of tools for accessing and using data in an OMERO server. Figures *OMERO.insight* and *OMERO.insight ImageViewer and Measurement Tool* present the user interface.

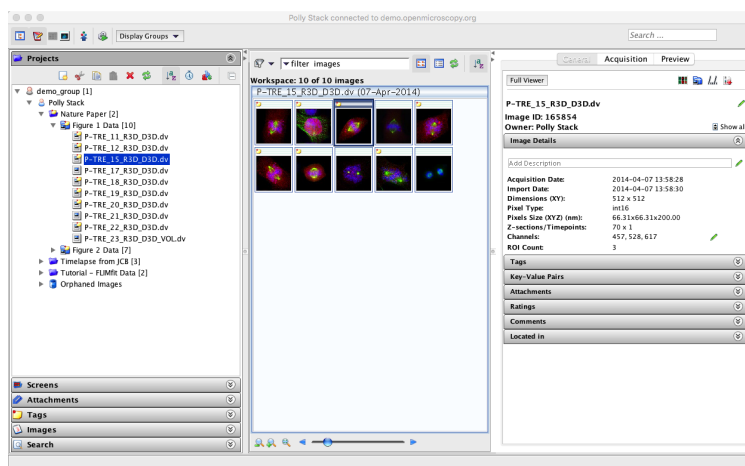


Fig. 3: OMERO.insight

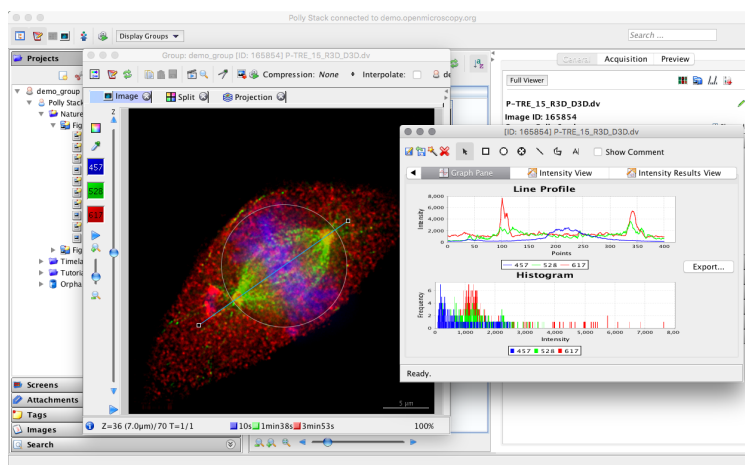


Fig. 4: OMERO.insight ImageViewer and Measurement Tool

The two main additional features of OMERO.insight which are not available as yet for OMERO.web are:

- Measurement Tool, a sub-application of ImageViewer that enables size and intensity measurements of defined regions-of-interest (ROIs)
- image import

We recommend to use OMERO.insight mainly for imports and concentrate on OMERO.web for all other workflows. See our [User guides](#) for a number of workflows showing how to import data using OMERO.insight and how to view, manage and export data using OMERO.web.

OMERO.importer

The OMERO.importer is part of the OMERO.insight client, but can also run as a stand-alone application. The OMERO.importer allows the import of proprietary image data files from a filesystem accessed from the user's computer to a running OMERO server. This tool uses a standard file browser to select the image files for import into an OMERO server.

The tool uses Bio-Formats for translation of proprietary file formats in preparation for upload to an OMERO.server. Visit [Supported Formats](#) for a detailed list of supported formats.

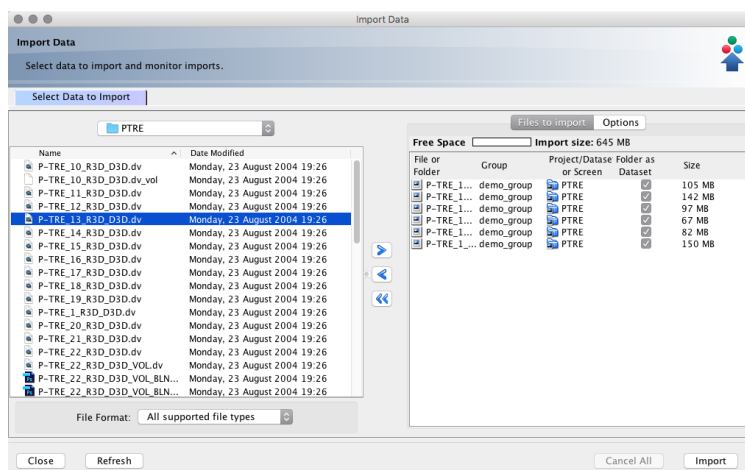


Fig. 5: OMERO.importer

OMERO.cli

The CLI (Command Line Interface) is a set of Python-based system administration, deployment and advanced user tools. Most of commands work remotely so that the CLI can be used as a client against an OMERO server. See *Command Line Interface as an OMERO client* for further information.

1.2.2 Command Line Interface as an OMERO client

The CLI is a set of Python based system-administration, deployment and advanced user tools. Most of commands work remotely so that the CLI can be used as a client against an OMERO server.

Note: The end of Windows support for OMERO.server means that the CLI is unsupported on this platform too.

Installation

Note: The CLI is currently untested on Windows but may be supported in the future.

Since OMERO 5.6, only Python 3 is supported. We assume that you have already installed Python 3.6 or higher. You can ensure that your python executable is correct with the `python --version` command.

We recommend installing client library `omero-py` and the CLI plugins in a Python virtual environment. You can create one using either `venv` or `conda` (preferred). If you opt for [Conda](#), you will need to install it first, see [miniconda](#) for more details.

Note: On Ubuntu 20.04, you may need to install `libssl-dev` before installing the CLI.

To install `omero-py` using `conda` (preferred):

```
conda create -n myenv -c conda-forge python=3.8 omero-py
conda activate myenv
```

Alternatively install `omero-py` using `venv` with Python 3.7 or higher:

```
python -m venv myenv
. myenv/bin/activate
pip install omero-py
```

The `omero` command is now available in the terminal where the environment has been activated:

```
omero login
```

If you install `omero-py` $\geq 5.8.0$ the CLI provides all functionalities except the `import` functionality.

The `import` functionality requires a supported version of [Java](#), and some JARs which are automatically downloaded the first time you do an import.

To install Java, go to [OMERO.server installation](#) and select the walkthrough corresponding to your OS.

omero-py < 5.8.0

If you are using an older version of `omero-py` you must download the JARs manually and place them under the `OMERODIR` directory:

1. download the `OMERO.server` zip from the [Downloads page](#)
2. unzip the zip file
3. set `$OMERODIR` to the unzipped directory:

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

The `import` functionality is now available:

```
omero import /path/to/image.tiff
```

Overview

Command line help

The CLI is divided into several commands which may themselves contain subcommands. You can investigate the various commands available using the `-h` or `--help` option:

```
$ omero -h
```

Again, you can use `-h` repeatedly to get more details on each of these sub-commands:

```
$ omero admin -h
$ omero admin start -h
```

The **omero help** command can be used to get info on other commands or options:

```
$ omero help admin      # same as omero admin -h
```

In addition to the CLI commands which can be listed using `omero help --list`, **omero help** can be used to retrieve information about the debug and env options:

```
$ omero help debug      # display help about debugging options
$ omero help env        # display help about environment variables
```

--all

Display the help for all available commands and options

--recursive

Recursively display the help of commands and/or options. This option can be used with either the **omero help** command or the `omero help --all` option:

```
$ omero help --all --recursive
$ omero help user --recursive
```

--list

Display a list of all available commands and subcommands

Command line workflow

There are three ways to use the command line tools:

1. By explicitly logging in to the server first i.e. by creating a session using the **omero login** command, e.g.:

```
$ omero login username@servername:4064
Password:
```

During login, a session is created locally on disk and will remain active until you logout or it times out. You can then call the desired command, e.g. the **omero import** command:

```
$ omero import image.tiff
```

2. By passing the session arguments directly to the desired command. Most commands support the same arguments as **omero login**:

```
$ omero -s servername -u username -p 4064 import image.tiff
Password:
```

The `--sudo` option is available to all commands accepting connection arguments. For instance to import data for user *username*:

```
$ omero import --sudo root -s servername -u username image.tiff
Password for owner:
```

3. By calling the desired command without login arguments. You will be asked to login:

```
$ omero import image.tiff
Server: [servername]
Username: [username]
Password:
```

Once you are done with your work, you can terminate the current session if you wish using the **omero logout** command:

```
$ omero logout
```

Visit [Manage sessions](#) to get a basic overview of how user sessions are managed.

See also:

[Advanced import scenarios](#)

[In-place import](#)

[OMERO.dropbox](#)

[Command Line Interface as an OMERO client](#)

Import images

The CLI import command allows you to import images to an OMERO.server from the command line, and is ideally suited for anyone wanting to use a shell-scripted or web-based front-end interface for importing. Based upon the same set of libraries as the standard importer, the command line version supports the same file formats and functions in much the same way. Visit [Supported Formats](#) for a detailed list of supported formats.

Overview

Visit [Overview](#) to get a basic overview of the CLI.

Installation

Visit [Installation](#) to install the CLI.

Import command

To import a file `image.tif`, use:

```
$ omero import image.tif
```

Some of the options available to the import command are:

-h, --help

-s SERVER, **-p** PORT, **-U** USERNAME, **-g** GROUPNAME

To avoid prompts for servername, port, username and group, use:

```
$ omero import -s SERVER -p PORT -u USER -g GROUP image.tif
```

-d DATASET_ID, **-r** SCREEN_ID, **-T** TARGET, **--target** TARGET

To import images into a Dataset:

```
$ omero import image.tif -d 2
$ omero import image.tif -T Dataset:id:2
$ omero import image.tif -T Dataset:name:Sample01
```

See [Import targets](#) for more information on import targets.

-n NAME, **--name** NAME

-x DESCRIPTION, **--description** DESCRIPTION

To change the name of an image and add a description:

```
$ omero import image.tif -n "control image1" -x "PBS control"
$ omero import image.tif --name image2 --description second_batch
```

--file FILE

File for storing the standard output from the Java process

--errs FILE

File for storing the standard error from the Java process

--logprefix DIR

Directory or file prefix for `-file` and `-errs`

--output TYPE

Set an alternative output style, for example:

```
$ omero import --output=yaml ...
```

Scanning folders prior to Import

-f

Display all the files that would be imported, then exit:

```
$ omero import -f image.tif
$ omero import -f images_folder
```

This will output a list of all the files which would be imported in groups separated by “#” comments. Note that this usage does not require a running server to be available.

--depth DEPTH

Set the number of directories to scan down for files (default: 4):

```
$ omero import --depth 7 images_folder
```

The above example changes the depth to 7 folders.

Bulk import configuration

--bulk YAML_FILE

To import a number of images with a similar configuration:

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

See [Bulk imports](#) for more information on bulk imports.

Managing performance of imports

--skip SKIP

Specify optional step to skip during import.

The import of very large datasets like High-Content Screening data or SPIM data can be time and resource consuming both at the client and at the server level. This option allows the disabling of some non-critical steps and thus faster import of such datasets. The caveat associated with its usage is that some elements are no longer generated at import time. Some of these elements, like thumbnails, will be generated at runtime during client access. Available options that can be skipped are currently:

all

Skip all optional steps described below

checksum

Skip checksum calculation on image files before and after transfer

This option effectively sets the `--checksum_algorithm` to use a fast algorithm, `File-Size-64`, that considers only file size, not the actual file contents.

minmax

Skip calculation of the minima and maxima pixel values

This option will also skip the calculation of the pixels checksum. Recalculating minima and maxima pixel values post-import is currently not supported. See [Calculation of minima and maxima pixel values](#) for more information.

thumbnails

Skip generation of thumbnails

Thumbnails will usually be generated when accessing the images post-import via the OMERO clients.

upgrade

Skip upgrade check for Bio-Formats

Example of usage:

```
$ omero import large_image --skip all
$ omero import large_image --skip minmax
```

Multiple import steps can be skipped by supplying multiple arguments:

```
$ omero import large_image --skip checksum --skip minmax
```

--parallel-fileset COUNT

Number of fileset candidates to import at the same time.

OMERO groups image files into *Filesets*. By default each fileset is imported one after another. This option attempts import of COUNT filesets at once. Even for single-file filesets it typically makes sense to use this option in conjunction with *--parallel-upload* so that upload of different filesets' files may proceed in parallel. For importing a single fileset containing many files this option will not help.

This is an *experimental* option. Too high a setting for COUNT may crash the import client or make the OMERO server unresponsive. Carefully read *Parallel import* before use.

--parallel-upload COUNT

Number of file upload threads to run at the same time.

By default files are uploaded one after another. Once a fileset's files are all on the server then it may commence subsequent import steps. It typically makes sense to set this to a value of at least the value for *--parallel-fileset*. Even if filesets are not imported in parallel this option can greatly speed the import of a fileset that consists of many small files.

This is an *experimental* option. Too high a setting for COUNT may crash the import client or make the OMERO server unresponsive. Carefully read *Parallel import* before use.

Checking performance

omero fs importtime finds out how long it took to import an existing fileset. Once the import is complete this command can estimate the wall-clock time taken for separate phases of the import process. Output is limited to what could be queried from the server easily. Specify the ID of a fileset to have its import time reported in a human-readable format.

--cache

Once import time has been determined for the specified fileset, also cache that information by annotating the fileset using a *map annotation* in the `openmicroscopy.org/omero/import/metrics` namespace. The cache will be used for future reports of that fileset's import time.

--summary

This report covers multiple filesets so do not provide a fileset ID. All data previously cached by the *--cache* option is queried then summarized in machine-readable CSV format.

Troubleshoot and report issues

--debug DEBUG

Set the debug level for the command line import output:

```
$ omero import images_folder --debug WARN
```

--report

Report emails to the OME team. This flag is mandatory for the `--upload` and `--logs` arguments.

--email EMAIL

Set the contact email to use when reporting errors. This argument should be used in conjunction with the `omero import --report` and `omero import --upload` or `omero import --logs` arguments.

--upload

Upload broken files and log file (if any) with report

The following command would import a broken image and upload it together with the import log if available in case of failure:

```
$ omero import broken_image --report --upload --email my.email@domain.com
```

--logs

Upload log file (if any) with report

The following command would import a broken image and upload only the import log if available in case of failure:

```
$ omero import broken_image --report --logs --email my.email@domain.com
```

Advanced import commands

--java-help

Display the help for the Java options of the import command

Java options can be passed after `--`

```
$ omero import image.tif -- --name=test --description=TestDescription
```

The above command will import the image “image.tif” with the name “test” into OMERO and with the OMERO description property set to “TestDescription”. Visit [Creating containers and annotations](#) to get a basic overview of how annotations can be created and linked to OMERO objects (object being an image, in this case).

--advanced-help

Display the advanced help for the import command, e.g.

```
$ omero import -- --advanced-help
```

Examples of usage,

To upload and remove the raw file from the local file-system after a successful import into OMERO, use:

```
$ omero import -- --transfer=upload_rm my_file.dv
```

As an OMERO administrator, to import images for other users, use:

```
$ omero login --sudo root -s servername -u username -g groupname
$ omero import image.tif
```

As an OMERO group owner, to import images for others, use:

```
$ omero login --sudo owner -s servername -u username -g groupname
$ omero import image.tif
```

Some advanced import options are described in the *In-place import* section. Visit *Manage sessions* to get a basic overview of how user sessions are managed.

Command Line Importer

The CLI import plugin calls the `ome.formats.importer.cli.CommandLineImporter` Java class. The Linux OMERO importer also includes an `importer-cli` shell script allowing calls to the importer directly from Java. Using `importer-cli` might look like this:

```
./importer-cli -s localhost -u user -w pass image.tif
```

To use the `ome.formats.importer.cli.CommandLineImporter` class from java on the command line you will also need to include a classpath to the required support jars. Please inspect the `importer-cli` script for an example of how to do this.

The Command Line Importer tool takes a number of mandatory and optional arguments to run. These options will also be displayed on the command line by passing no arguments to the importer:

Import any number of files into an OMERO instance.

If "-" is the only path, a list of files or directories is read from standard in. Directories will be searched for all valid imports.

Session arguments:

Mandatory arguments for creating a session are 1- either the OMERO server hostname, username and password or 2- the OMERO server hostname and a valid session key.

```
-s SERVER      OMERO server hostname
-u USER       OMERO username
-w PASSWORD    OMERO password
-k KEY         OMERO session key (UUID of an active session)
-p PORT        OMERO server port (default: 4064)
```

Naming arguments:

All naming arguments are optional

```
-n NAME          Image or plate name to use
-x DESCRIPTION   Image or plate description to use
--name NAME      Image or plate name to use
--description DESCRIPTION Image or plate description to use
```

Optional arguments:

```
-h              Display this help and exit
-f              Display the used files and exit
-c              Continue importing after errors
-l READER_FILE  Use the list of readers rather than the default
```

(continues on next page)

(continued from previous page)

<code>-d DATASET_ID</code>	OMERO dataset ID to import image into
<code>-r SCREEN_ID</code>	OMERO screen ID to import plate into
<code>-T TARGET</code>	target for imports
<code>--report</code>	Report errors to the OME team
<code>--upload</code>	Upload broken files and log file (if any) with <code>--report</code> . Required <code>--report</code>
<code>--logs</code>	Upload log file (if any) with report. Required <code>--report</code>
<code>--email EMAIL</code>	Email for reported errors. Required <code>--report</code>
<code>--debug LEVEL</code>	Turn debug logging on (optional level)
<code>--annotation-ns ANNOTATION_NS</code>	Namespace to use for subsequent annotation
<code>--annotation-text ANNOTATION_TEXT</code>	Content for a text annotation
<code>--annotation-link ANNOTATION_LINK</code>	Comment annotation ID to link all images to

Examples:

```
$ importer-cli -s localhost -u user -w password -d 50 foo.tiff
$ importer-cli -s localhost -u user -w password -d Dataset:50 foo.tiff
$ importer-cli -f foo.tiff
$ importer-cli -s localhost -u username -w password -d 50 --debug ALL foo.tiff
```

For additional information, see:
<https://docs.openmicroscopy.org/latest/omero/users/cli/import.html>
 Report bugs at <https://www.openmicroscopy.org/forums>

See also:*Advanced import scenarios**In-place import**OMERO.dropbox**Command Line Interface as an OMERO client***Import targets**

The CLI import options `-d` or `-r` can be used to specify, respectively, the import target Dataset or Screen by ID. The `-T`, `--target` option adds more ways of specifying the import target.

The general form of the target argument is:

```
<action or Class>[:<discriminator>]:<pattern>
```

where the discriminator is optional. Thus a target must contain one or two colons. Any further colons will be read as part of the pattern. If the discriminator is omitted a default will be used depending on the action or Class. Currently the following actions and classes are supported: Dataset, Screen and regex.

Importing to a Dataset or Screen

For Dataset and Screen the currently supported discriminators are name and id. If the discriminator is omitted the default used is id. So:

```
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:id:2
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:2
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -d 2
```

will all have the same effect of importing the image to the Dataset with ID 2.

The name discriminator can be used to select the target by name, and so:

```
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:name:Sample01
```

will import the image to the Dataset with name Sample01. If more than one Dataset exists with the specified name the most recently created will be used. If no Dataset exists with the specified name a new Dataset will be created for the import.

The choice of Dataset can be specified by adding a qualifying character to the discriminator: + to use the most recent, - to use the oldest, % to only import if there is a unique target or @ to create a new container even if one with the correct name already exists.

For example:

```
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:+name:Samples
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:-name:Samples
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:%name:Samples
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:@name:Samples
```

The first case is equivalent to the previous example, the most recent Dataset will be used. In the second case the oldest Dataset will be used. In the third case the import should fail if multiple datasets with that name exist. In the first three cases a new Dataset will be created if none exists. In the last case a new Dataset should be created even if one or more already exist.

If the name contains spaces or other characters that cannot be used on the command line the pattern should be enclosed in quotes:

```
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:name:"New Dataset"
```

To import a plate to a Screen target the same syntax can be used as in all the examples above, for example:

```
$ omero import ~/images/bd-pathway/2015-12-01_000/ -T Screen:+name:Pathway
```

Importing to a Dataset inside a specific Project

To import an image into a Dataset contained in a specific Project, use:

```
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Project:name:"Proj1"/Dataset:name:
↪ "New Dataset"
```

The above command will create a new Project Proj1 and link the Dataset New Dataset to it, except in case a Project named Proj1 already exists. Then, the Dataset named New Dataset will be linked to this existing Project.

Analogically, a new Dataset named `New Dataset` will be created for the import of the image and linked to the Project `Proj1`, except in case a Dataset `New Dataset` already exists. Then, the existing Dataset will be used for the import of the image and linked to Project `Proj1`.

Warning: If a Dataset named `New Dataset` already exists and has been linked prior to your import to some other Project (for example `ProjP`), this existing Dataset will be used as the target Dataset container and will be linked both to `ProjP` and `Proj1` after the import.

Importing using regular expressions

The local path of the file to be imported can be used to specify the target Dataset or Screen using a regular expression using the action `regex`. For this action the only discriminator is `name` and if the discriminator is omitted the qualified form of this `+name` will be used. The sequence `(?<Container1>.*?)` is a named-capturing group used to specify the Dataset or Screen name in the regular expression, the specific name `Container1` must be used here. For example:

```
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T "regex:^. *images/(?<Container1>.*?)"
```

would use a Dataset with name being the path following `images/`, in this case `dv`.

The name discriminator can be explicitly used and, as in the previous section, also qualified:

```
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T "regex:+name:^. *images/(?<Container1>.*?)"
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T "regex:-name:^. *images/(?<Container1>.*?)"
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T "regex:%name:^. *images/(?<Container1>.*?)"
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T "regex:@name:^. *images/(?<Container1>.*?)"
```

These each work in the same way as the previous Dataset examples.

In some cases the importable files may be in nested directories, this is often the case with plates and some multi-image formats. A regular expression can be used to pick a higher level directory as the Screen or Dataset name. For example, if several BD Pathway HCS files are under the following paths:

```
~/images/bd-pathway/week-1/2015-12-01_000/
~/images/bd-pathway/week-2/2015-12-09_000/
~/images/bd-pathway/week-2/2015-12-11_000/
```

and the intended Screens for the import are `week-1` and `week-2` then the following could be used:

```
$ omero import ~/images/bd-pathway/ -T "regex:+name:^. *bd-pathway/(?<Container1>[^/]+)/.*"
```

which would import one Plate into the Screen `week-1` and two Plates into the Screen `week-2`, creating those Screens if necessary.

A useful way of determining the nested structure to help in constructing regular expressions is the option `-f` which displays the used files but does not import them:

```
$ omero import -f ~/images/bd-pathway/week-1
...
```

(continues on next page)

(continued from previous page)

```

2016-03-30 15:58:56,574 701      [    main] INFO      ome.formats.importer.
↳ ImportCandidates - 59 file(s) parsed into 1 group(s) with 1 call(s) to setId in 92ms.↳
↳ (99ms total) [0 unknowns]
#=====
# Group: /Users/colin/images/bd-pathway/week-1/2009-05-01_000/Experiment.exp SPW: true↳
↳ Reader: loci.formats.in.BDReader
/Users/colin/images/bd-pathway/week-1/2009-05-01_000/Experiment.exp
/Users/colin/images/bd-pathway/week-1/2009-05-01_000/20X NA 075 Olympus Confocal.geo
...
/Users/colin/images/bd-pathway/week-1/2009-05-01_000/Well D11/DsRed - Confocal - n0000000.
↳ tif
/Users/colin/images/bd-pathway/week-1/2009-05-01_000/Well D11/DsRed - Confocal - n0000001.
↳ tif
/Users/colin/images/bd-pathway/week-1/2009-05-01_000/Well D11/DsRed - Confocal - n0000002.
↳ tif
/Users/colin/images/bd-pathway/week-1/2009-05-01_000/Well D11/Transmitted Light -↳
↳ n0000000.tif

```

which shows that all the files for one particular Plate from the example above are under:

```
/Users/colin/images/bd-pathway/week-1/2009-05-01_000/
```

For more information on the regular expression syntax that can be used in templates see: [java.util.regex.Pattern documentation](#).

Importing to targets across groups

Currently, in all the above cases the import target must be in the user's current group for the import to succeed. It is hoped that this limitation can be removed in a later version of OMERO. This is also pertinent if the target is likely to be created as it will be created in the current group, which may not be the group intended.

If no group is specified by using the `omero login -g` option as part of the import, the current group will be dependent on the user's login status:

- If the user is currently logged in then their current group will be the one they are logged in to.
- If the user is logged out but has active sessions then the most recent session will be used to connect and that will determine the current group.
- If the user is logged out and has no active sessions then the current group will be their default group.

If the user knows which group the import target is in, or needs to be created in, then one of the following methods can be used to ensure the target group is the current group for the import:

- Explicitly log in using the `omero login -g` option before running the import command:

```
$ omero login -g group_name
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:2
```

- Provide the `omero login -g` option as part of the import command:

```
$ omero import -g group_name ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:2
```

- Use `omero sessions group` to switch group before running the import command:

```
$ omero sessions group 51
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:2
```

- Use the `omero login -k` option to reconnect to an active session for the target group:

```
$ omero login -k c41a6f78-ba6e-4caf-aba3-a94378d5484c
$ omero import ~/images/dv/SMN10ul03_R3D_D3D.dv -T Dataset:2
# or alternatively
$ omero import -k c41a6f78-ba6e-4caf-aba3-a94378d5484c ~/images/dv/SMN10ul03_R3D_
↪D3D.dv -T Dataset:2
```

The session ID can be found using the `omero sessions list` command.

For further information on the commands `omero login` and `omero sessions` see [Manage sessions](#).

Note: The `omero login -g` option requires the group name as its argument, while the `omero sessions group` subcommand uses either the group ID or the group name.

See also:

[Advanced import scenarios](#)

[In-place import](#)

[OMERO.dropbox](#)

[Command Line Interface as an OMERO client](#)

Bulk imports

The CLI import option `--bulk` specifies a configuration file that can be used to perform a batch of imports with the same or similar options. The file is written in a simple YAML syntax and can be named whatever you would like. It does not need to be placed in the folder from which the OMERO commands are run.

A minimal YAML file might look like:

```
---
path: "my-files.txt"
```

Assuming that `my-files.txt` is a list of files such as

```
fileA
fileB
directoryC
```

this is equivalent to:

```
$ omero import -k --transfer=ln_s fileA fileB directoryC
```

where the files `fileA` and `fileB` and all the files of `directoryC` will be imported.

Bulk-only options

Path

The `path` key specifies a file from which *each individual line* will be processed as a separate import. In the simplest case, a single file is placed per line as above. For more complex usages, `path` can point to a tab-separated value (TSV) or a comma-separated value (CSV) file where each field will be interpreted based on `columns`.

Columns

A fairly regular requirement in importing many files is that for each file a similar but slightly different configuration is needed. This can be accomplished with the `columns` key. It specifies how each of the separated fields of the `path` file should be interpreted.

For example, a `bulk.yml` file specifying:

```
---
path: "files.tsv"
columns:
- name
- path
```

along with a `files.tsv` of the form:

```
import-1    fileA
import-2    fileB
```

would match the two calls:

```
$ omero import --name import-1 fileA
$ omero import --name import-2 fileB
```

but in a single call. The same could be achieved with this CSV file:

```
import-1,fileA
import-2,fileB
```

Other options like `target` can also be added as a separate field:

```
Dataset:name:training-set  import-1    fileA
Dataset:name:training-set  import-2    fileB
Dataset:name:test-set-001  import-3    fileC
```

by defining `columns` in your `bulk.yml` as:

```
columns:
- target
- name
- path
```

which will create the named datasets if they do not exist. See [Import targets](#) for more information on import targets and see below for more examples of options you can use.

Include

The `include` key specifies another bulk YAML file that should be included in the current processing. For example, if there is a global configuration file `omero-imports.yml` that all users should use, such as:

```
---
checksum_algorithm: "File-Size-64"
exclude: "clientpath"
transfer: "ln_s"
```

then users can make use of this configuration by adding the following line to their `bulk.yml` file:

```
include: /etc/omero-imports.yml
```

Dry_run

The `dry_run` key can either be set to `true` in which case no import will occur, and only the potential actions will be shown, or additionally it can be set to a file path of the form `my_import_%s.sh` where `%s` will be replaced by an number and a file with the given name will be written out. Each of these scripts can then be used independently.

Other options

Otherwise, all the regular options from the CLI are available for configuration via `--bulk`:

- `checksum_algorithm` for faster processing of large files
- `continue` for processing all files even if one errors
- `exclude` for skipping files that have already been imported
- `parallel_fileset` for concurrent imports
- `parallel_upload` for concurrent uploads
- `target` for placing imported images into specific containers
- `transfer` for alternative methods of shipping files to the server

See [Import images](#) for more information.

Export images

The CLI export command allows you to export data in XML and OME-TIFF formats from an OME-RO server using the command line.

Overview

Visit [Overview](#) to get a basic overview of the CLI.

Installation

Visit [Installation](#) to install the CLI.

Export command

Currently the export command only supports OME-TIFF, respectively XML.

To export an image as OME-TIFF as file `image.tif`, use:

```
$ omero export --file image.tif Image:<id>
```

To export its metadata as file `image.xml`, use:

```
$ omero export --file image.xml --type XML Image:<id>
```

Some of the options available to the export command are:

--iterate Dataset:<id>

Iterate over an object and write individual objects to the directory named by `--file` (EXPERIMENTAL, the only supported object is Dataset:<id>)

```
$ omero export --file output-dir --iterate Dataset:<id>
```

Manage sessions

The **omero sessions** plugin manage user sessions stored locally on disk. Several sessions can be active simultaneously, but only one will be used for a single invocation of *omero*:

```
$ omero sessions -h
```

Login

The **omero login** command is a shortcut for the **omero sessions login** subcommand which creates a connection to the server. If no argument is specified, the interface will ask for the connection credentials:

```
$ omero login
Previously logged in to localhost:4064 as root
Server: [localhost:4064]
Username: [root]
Password:
```

Some of the options available to the **omero login** command are:

connection

Pass a connection string under the form `[USER@]SERVER[:PORT]` to instantiate a connection:

```
$ omero login username@servername
Password:
$ omero login username@servername:14064
Password:
```

-s SERVER, --server SERVER

Set the name of the server to connect to:

```
$ omero login -s servername
Username: [username]
```

-u USER, --user USER

Set the name of the user to connect as:

```
$ omero login -u username -s servername
Password:
```

-p PORT, --port PORT

Set the port to use for connection. Default: 4064:

```
$ omero login -u username -s servername -p 14064
Password:
```

-g GROUP, --group GROUP

Set the group to use for initializing a connection:

```
$ omero login -u username -s servername -g my_group
Password:
```

-k KEY, --key KEY

Use a valid session key to join an existing connection.

This option only requires a server argument:

```
$ omero login servername -k 22fccb8b-d04c-49ec-9d52-116a163728ca
```

-w PASSWORD, --password PASSWORD

Set the password to use for the connection. Since 5.4.1, the password can be set using the `OMERO_PASSWORD` environment variable. The variable will be ignored if `-w` or `--password` is used.

--sudo ADMINUSER|GROUPOWNER

Create a connection as another user.

The sudo functionality is available to administrators as well as group owners

```
$ omero login --sudo root -s servername -u username -g groupname
Password for root:
$ omero login --sudo owner -s servername -u username -g groupname
Password for owner:
```

Multiple sessions

Stored sessions can be listed using the **omero sessions list** command:

```
$ omero sessions list
Server      | User | Group           | Session                                     | Active |
└─┘Started
-----+-----+-----+-----+-----+-----+
└─┘-----
localhost | test | read-annotate-2 | 22fccb8b-d04c-49ec-9d52-116a163728ca | Logged in |
└─┘Fri Nov 23 14:55:25 2012
localhost | root | system          | 1f800a16-1dc2-407a-8a85-fb44005306be | True      |
└─┘Fri Nov 23 14:55:18 2012
(2 rows)
```

Session keys can then be reused to switch between stored sessions using the **omero login -k** option:

```
$ omero sessions login -k 22fccb8b-d04c-49ec-9d52-116a163728ca
Server: [localhost]
Joined session 1f800a16-1dc2-407a-8a85-fb44005306be (root@localhost:4064).
$ omero sessions list
Server      | User | Group           | Session                                     | Active |
└─┘Started
-----+-----+-----+-----+-----+-----+
└─┘-----
localhost | test | read-annotate-2 | 22fccb8b-d04c-49ec-9d52-116a163728ca | True      |
└─┘Fri Nov 23 14:55:25 2012
localhost | root | system          | 1f800a16-1dc2-407a-8a85-fb44005306be | Logged in |
└─┘Fri Nov 23 14:55:18 2012
(2 rows)
```

Sessions directory

By default sessions are saved locally on disk under the OMERO user directory located at `~/omero/sessions`. The location of the current session file can be retrieved using the **omero sessions file** command:

```
$ omero sessions file
/Users/ome/omero/sessions/localhost/root/aec828e1-79bf-41f3-91e6-a4ac76ff1cd5
```

To customize the OMERO user directory, use the `OMERO_USERDIR` environment variable:

```
$ export OMERO_USERDIR=/tmp/omero_dir
$ omero login root@localhost:4064 -w omero
Created session bf7b9fee-5e3f-40fa-94a6-1e23ceb43dbd (root@localhost:4064). Idle
└─┘timeout: 10.0 min. Current group: system
$ omero sessions file
/tmp/omero_dir/omero/sessions/localhost/root/bf7b9fee-5e3f-40fa-94a6-1e23ceb43dbd
$ omero logout
```

If you want to use a custom directory for sessions exclusively, use the `OMERO_SESSIONDIR` environment variable:

```
$ export OMER_SESSIONDIR=/tmp/my_sessions
$ omero login root@localhost:4064 -w omero
Created session bf7b9fee-5e3f-40fa-94a6-1e23ceb43dbd (root@localhost:4064). Idle
↪ timeout: 10.0 min. Current group: system
$ omero sessions file
/tmp/my_sessions/localhost/root/bf7b9fee-5e3f-40fa-94a6-1e23ceb43dbd
$ omero logout
```

Note: The `OMERO_SESSION_DIR` environment variable introduced in 5.1.0 to specify a custom sessions directory is deprecated in 5.1.1 and above in favor of `OMERO_SESSIONDIR`.

If you have been using `OMERO_SESSION_DIR` and want to upgrade your custom sessions directory without losing locally stored sessions:

- either set `OMERO_SESSIONDIR` to point at the same location as `OMERO_SESSION_DIR/omero/sessions`
- or move all local sessions stored under the `OMERO_SESSION_DIR/omero/sessions` directory under the `OMERO_SESSION_DIR` directory and replace `OMERO_SESSION_DIR` by `OMERO_SESSIONDIR`.

Switching current group

The **sessions group** command can be used to switch the group of your current session:

```
$ omero group list          # list your groups
$ omero sessions group 2    # switch to group by ID or Name
```

Creating containers and annotations

The **omero obj** command allows users to create and update OMERO objects. A complete *Glossary of all OMERO Model Objects* is available for reference.

This command can be used to create containers, i.e. projects, datasets, screens and folders. It can also be used to create annotations, and, combined with the **omero upload** command, file annotations. These annotations can then be attached to containers or imported images and plates. This page gives a few examples of some simple but fairly common workflows.

Creating containers

Create a dataset with a name:

```
$ omero obj new Dataset name=NewDVSet
Dataset:51
```

And then update that dataset to add a description:

```
$ omero obj update Dataset:51 description='A dataset for new DV images'
Dataset:51
```

Create a screen with a name and description:

```
$ omero obj new Screen name=Screen001 description='A short description'
```

To create a project/dataset hierarchy a link must be created between the two containers:

```
$ omero obj new Project name=NewImages
Project:101
$ omero obj new ProjectDatasetLink parent=Project:101 child=Dataset:51
ProjectDatasetLink:221
```

If you are comfortable using the command line then you can capture the command outputs to feed in to other commands, for example:

```
$ dataset=$(omero obj new Dataset name=dataset-1)
$ project=$(omero obj new Project name=project-1)
$ omero obj new ProjectDatasetLink parent=$project child=$dataset
ProjectDatasetLink:222
```

Creating and attaching annotations

Create a comment annotation and attach it to a dataset:

```
$ omero obj new CommentAnnotation textValue='Hello World!'
CommentAnnotation:2
$ omero obj new DatasetAnnotationLink parent=Dataset:51 child=CommentAnnotation:2
DatasetAnnotationLink:2
```

Upload a file and then use it as file annotation on an image:

```
$ omero upload analysis.csv
OriginalFile:275
$ omero obj new FileAnnotation file=OriginalFile:275
FileAnnotation:5
$ omero obj new ImageAnnotationLink parent=Image:51 child=FileAnnotation:5
ImageAnnotationLink:2
```

Manage tags

The **omero tag** subcommands manage the creation, linking and listing of tag annotations. All subcommands can be listed using the **-h** option:

```
$ omero tag -h
```

Create tags

To create a new tag annotation, use the **omero tag create** command:

```
$ omero tag create
Please enter a name for this tag: my_tag
```

To create a tag set containing two existing tags of known identifiers 1259 and 1260, use the **omero tag createset** command:

```
$ omero tag createset --tag 1259 1260
Please enter a name for this tag set: my_tag_set
```

For both tags and tag sets, the name and an optional description can be passed using the **--name** and **--desc** options:

```
$ omero tag create --name my_tag --desc 'description of my_tag'
$ omero tag createset --tag 1259 1260 --name my_tag_set --desc 'description of my_tag_set'
↪ '
```

List tags

To list all the tags owned by the current user, use the **omero tag list** command:

```
$ omero tag list
+- 1261: 'my_tag_set'
| \
| +- 1259: 'my_tag'
| +- 1260: 'my_tag_2'
+- 1264: 'my_tag_set_2'
| \
| +- 1260: 'my_tag_2'
| +- 1263: 'my_tag_4'

Orphaned tags:
> 1262: 'my_tag_3'
```

To list all the tag sets owned by the current user, use the **omero tag listsets** command:

```
$ omero tag listsets
-----|-----
↪ -----
ID      |Name
-----|-----
↪ -----
1261    |my_tag_set
1264    |my_tag_set_2
-----|-----
↪ -----
```


Link tags

Tags can be linked to objects on the server using the **omero tag link** command. The object must be specified as `object_type:object_id`. To link the tag of identifier 1260 to the Image of identifier 1000, use:

```
$ omero tag link Image:1000 1260
```

Delete tags

Tags can be deleted using the **omero delete** command. The tag or tag set must be specified as `TagAnnotation:tag_id`. To delete tag 123 use:

```
$ omero delete TagAnnotation:123
```

By default the tags within a tag set will not be deleted with the tag set. To delete any included tags use the *omero delete --include* option:

```
$ omero delete TagAnnotation:123 --include TagAnnotation
```

See also:

Deleting objects

Deleting objects

The **omero delete** command deletes objects. Further help is available using the `-h` option:

```
$ omero delete -h
```

This command will remove entire graphs of objects based on the IDs of the topmost objects. The command can be modified to include the deletion of objects that would, by default, be excluded or exclude objects that would, by default, be included using the *omero delete --include* and *omero delete --exclude* options.

Additionally, objects of the three annotation types, *FileAnnotation*, *TagAnnotation* and *TermAnnotation* are not deleted by default when the objects to which they are linked are deleted.

It is also possible to delete objects lower in the hierarchy by specifying the type and ID of a topmost object and the type of the lower object. For instance, deleting all of the images under a given project.

By default the command confirms the deletion of the target objects but it can also provide a detailed report of all the deleted objects via an *omero delete --report* option. An *omero delete --dry-run* option can be used to report on what objects would be deleted without actually deleting them.

Examples

Basic delete

```
$ omero delete OriginalFile:101
$ omero delete Project:51
```

In the first line, the original file with ID 101 will be deleted. In the second, the project with ID 51 will be deleted including any datasets inside only that project and any images that are contained within deleted datasets only. Note that any linked file, tag or term annotations will not be deleted.

Deleting multiple objects

Multiple objects can be specified with each type being followed by an ID or a comma-separated list of IDs. The order of objects or IDs is not significant, thus all three calls below are identical in deleting project 51 and datasets 53 and 54.

```
$ omero delete Project:51 Dataset:53,54
$ omero delete Dataset:54,53 Project:51
$ omero delete Dataset:53 Project:51 Dataset:54
```

To delete a number of objects with sequentially numbered IDs a hyphen can be used to specify an ID range. This form can also be mixed with comma-separated IDs.

```
$ omero delete Project:51 Dataset:53-56 --force
$ omero delete Dataset:53-56,65,101-105,201,202 --force
```

When deleting multiple objects in a single command, if one object cannot be deleted then the whole command will fail and none of the specified objects will be deleted.

The `omero delete --dry-run` option can be useful as a check before trying to delete large numbers of objects. If specifying objects with a range, it is best to pass either `omero delete --dry-run` or `omero delete --force`.

Note: If no flag is passed, the command will default to `omero delete --dry-run` and warn that this behavior is deprecated. Future versions will default to `omero delete --force`.

Deleting lower level objects

To delete objects below a specified top-level object the following form of the object specifier is used.

```
$ omero delete Project/Dataset/Image:51
```

Here the all of images under the project 51 would be deleted. It is not necessary to specify intermediate objects in the hierarchy and so:

```
$ omero delete Project/Image:51
```

would have the same effect as the call above. Links can also be deleted and so:

```
$ omero delete Project/DatasetImageLink:51 Dataset/DatasetImageLink:53
```

would effectively orphan all images under project 51 and dataset 53 that are not also under other datasets.

Including and excluding objects

--include

Include linked objects that would not ordinarily be deleted:

```
$ omero delete Image:51 --include FileAnnotation,TagAnnotation,TermAnnotation
```

As mentioned above these three annotation types are not deleted by default and so this call overrides that default by including any of the three annotation types in the delete:

```
$ omero delete Image:51 --include Annotation
```

This call would also delete any annotation objects linked to the image.

--exclude

Exclude linked objects that would ordinarily be deleted:

```
$ omero delete Project:51 --exclude Dataset
```

This will delete project 51 but not any datasets contained in that project.

The two options can be used together:

```
$ omero delete Project/Dataset:53 --exclude Image --include FileAnnotation
```

This will delete any datasets under project 53, that are not otherwise contained elsewhere, excluding any images in those datasets but including any file annotations linked to the deleted datasets. In this case the images that are not otherwise contained in datasets will be orphaned.

For an example on deleting tags directly see [Delete tags](#).

Further options

--ordered

Delete the objects in the order specified.

Normally all of the specified objects are grouped into a single delete command. However, each object can be deleted separately and in the order given. Thus:

```
$ omero delete Dataset:53 Project:51 Dataset:54 --ordered
```

would be equivalent to making three separate calls:

```
$ omero delete Dataset:53
$ omero delete Project:51
$ omero delete Dataset:54
```

--report

Provide a detailed report of what is deleted:

```
$ omero delete Project:502 --report
...
omero.cmd.Delete2 Project 502... ok
Steps: 3
Elapsed time: 0.597 secs.
Flags: []
Deleted objects
Dataset:603
DatasetImageLink:303
Project:503
ProjectDatasetLink:353
Channel:203
Image:503
LogicalChannel:203
```

(continues on next page)

(continued from previous page)

```
OriginalFile:460,459
Pixels:253
Fileset:203
FilesetEntry:253
FilesetJobLink:264,265,262,263,261
IndexingJob:315
JobOriginalFileLink:303
MetadataImportJob:312
PixelDataJob:313
ThumbnailGenerationJob:314
UploadJob:311
StatsInfo:72
```

--dry-run

Run the command and report success or failure but do not delete the objects. This can be combined with the `omero delete --report` to provide a detailed confirmation of what would be deleted before running the delete itself.

--force

Delete multiple objects in a single command. Both comma-separated lists and ranges of IDs using a hyphen will work:

```
$ omero delete Project:51 Dataset:53-56,65,101-105 --force
```

The command will fail and no objects will be deleted if any of the specified objects cannot be deleted.

Moving objects between groups

Warning: Data does not need to be assigned to a group where the data owner is a member, and administrators may wish to change the ownership of data or move it between groups in several steps of a larger workflow. However, it is generally expected that data should end up in a group where the data owner is a member, so that they can view their data in the OMERO clients.

Who may move data

- a full administrator
- a *restricted administrator* with *Chgrp* privilege
- the owner of the data *if* they are a member of the target group

How to move data

- Using CLI: See below
- Using OMERO.web: See [User guides](#)

The **omero chgrp** command moves objects between groups. Further help is available using the **-h** option:

```
$ omero chgrp -h
```

This command will move entire graphs of objects based on the IDs of the topmost objects. The command can be modified to include the movement of objects that would, by default, be excluded or exclude objects that would, by default, be included using the *omero chgrp --include* and *omero chgrp --exclude* options.

It is also possible to move objects lower in the hierarchy by specifying the type and ID of a topmost object and the type of the lower object. For instance, moving all of the images under a given project.

By default the command confirms the movement of the target objects but it can also provide a detailed report of all the moved objects via an *omero chgrp --report* option. An *omero chgrp --dry-run* option can be used to report on what objects would be moved without actually moving them.

Examples

Basic move

```
$ omero chgrp 5 OriginalFile:101
$ omero chgrp Group:5 Project:51
$ omero chgrp ExperimenterGroup:5 Project:51
$ omero chgrp lab_group Project:51
```

In the first line, the original file with ID 101 will be moved to the group with ID 5. In the second and third, project 51 will be moved to group 5 including any datasets inside only that project and any images that are contained within moved datasets only. If group 5 is named 'lab_group' then the last line will have the same effect as the previous two. Note that any linked annotations will also be moved.

Moving multiple objects

Multiple objects can be specified with each type being followed by an ID or a comma-separated list of IDs. The order of objects or IDs is not significant, thus all three calls below are identical in moving project 51 and datasets 53 and 54 to group 5.

```
$ omero chgrp 5 Project:51 Dataset:53,54
$ omero chgrp 5 Dataset:54,53 Project:51
$ omero chgrp 5 Dataset:53 Project:51 Dataset:54
```

To move a number of objects with sequentially numbered IDs a hyphen can be used to specify an ID range. This form can also be mixed with comma-separated IDs.

```
$ omero chgrp 5 Project:51 Dataset:53-56
$ omero chgrp 5 Dataset:53-56,65,101-105,201,202
```

Note: When moving multiple objects in a single command, if one object cannot be moved then the whole command will fail and none of the specified objects will be moved. The `omero chgrp --dry-run` option can be useful as a check before trying to move large numbers of objects.

Moving lower level objects

To move objects below a specified top-level object the following form of the object specifier is used.

```
$ omero chgrp 5 Project/Dataset/Image:51
```

Here the all of images under the project 51 would be moved. It is not necessary to specify intermediate objects in the hierarchy and so:

```
$ omero chgrp 5 Project/Image:51
```

would have the same effect as the call above.

Including and excluding objects

--include

Linked objects that would not ordinarily be moved can be included in the move using the `--include` option:

```
$ omero chgrp 5 Image:51 --include Annotation
```

This call would move any annotation objects linked to the image.

--exclude

Linked objects that would ordinarily be moved can be excluded from the move using the `--exclude` option:

```
$ omero chgrp 5 Project:51 --exclude Dataset
```

This will move project 51 but not any datasets contained in that project.

The two options can be used together:

```
$ omero chgrp 5 Project/Dataset:53 --exclude Image --include FileAnnotation
```

This will move any datasets under project 53, that are not otherwise contained elsewhere, excluding any images in those datasets but including any file annotations linked to the moved datasets. In this case the images that are not otherwise contained in datasets will be orphaned.

Further options

--ordered

Move the objects in the order specified.

Normally all of the specified objects are grouped into a single move command. However, each object can be moved separately and in the order given. Thus:

```
$ omero chgrp 5 Dataset:53 Project:51 Dataset:54 --ordered
```

would be equivalent to making three separate calls:

```
$ omero chgrp 5 Dataset:53
$ omero chgrp 5 Project:51
$ omero chgrp 5 Dataset:54
```

--report

Provide a detailed report of what is moved:

```
$ omero chgrp 5 Project:502 --report
```

--dry-run

Run the command and report success or failure but does not move the objects. This can be combined with the `omero chgrp --report` to provide a detailed confirmation of what would be moved before running the move itself.

Changing ownership of objects

Warning: Data does not need to be assigned to a group where the data owner is a member, and administrators may wish to change the ownership of data or move it between groups in several steps of a larger workflow. However, it is generally expected that data should end up in a group where the data owner is a member, so that they can view their data in the OMERO clients.

Who may change ownership of data

- a full administrator
- a *restricted administrator* with *Chown* privilege
- an owner of the group that the data is in *if* the target user is a member of the group

How to change ownership of data

The **omero chown** command transfers objects to the ownership of a different user. Further help is available using the `-h` option:

```
$ omero chown -h
```

The **omero chown** command can transfer entire graphs of objects based on the IDs of the topmost objects. The command can be modified to include the transfer of objects that would, by default, be excluded or exclude objects that would, by default, be included using the `omero chown --include` and `omero chown --exclude` options.

It is also possible to transfer objects lower in the hierarchy by specifying the type and ID of a topmost object and the type of the lower object. For instance, transferring all of the images under a given project.

All the data of a given user can be transferred using the **omero chown** command. This is useful when somebody leaves a lab to move on to another project or institution and their previous work is to be curated or continued by a colleague. This feature has to be considered as advanced and might be slow and demanding of CPU resources in cases of complex data.

By default the command confirms the transfer of the target objects but it can also provide a detailed report of all the transferred objects via an `omero chown --report` option. An `omero chown --dry-run` option can be used to report on which objects' ownership would change without actually transferring them.

Examples

Basic transfer of ownership

```
$ omero chown 5 OriginalFile:101
$ omero chown User:5 Project:51
$ omero chown Experimenter:5 Project:51
$ omero chown jane Project:51
```

In the first line, the ownership of original file with ID 101 will be transferred to the user with ID 5. In the second and third, the ownership of project 51 will be transferred including any datasets inside only that project and any images that are contained within transferred datasets only, as long as all the mentioned objects (project, datasets and images) are originally owned by one user. If user 5 is named 'jane' then the last line will have the same effect as the previous two. Note that any linked annotations will be transferred depending on the permission level of the group in which the data and users are in.

Transferring multiple objects

Multiple objects can be specified with each type being followed by an ID or a comma-separated list of IDs. The order of objects or IDs is not significant, thus all three calls below are identical in transferring ownership of project 51 and datasets 53 and 54 to user 5.

```
$ omero chown 5 Project:51 Dataset:53,54
$ omero chown 5 Dataset:54,53 Project:51
$ omero chown 5 Dataset:53 Project:51 Dataset:54
```

To transfer a number of objects with sequentially numbered IDs a hyphen can be used to specify an ID range. This form can also be mixed with comma-separated IDs.

```
$ omero chown 5 Project:51 Dataset:53-56
$ omero chown 5 Dataset:53-56,65,101-105,201,202
```

Note: When transferring multiple objects in a single command, if one object cannot be transferred then the whole command will fail and none of the specified objects will be transferred. The `omero chown --dry-run` option can be useful as a check before trying to move large numbers of objects.

Transferring lower level objects

To transfer objects below a specified top-level object the following form of the object specifier is used.

```
$ omero chown 5 Project/Dataset/Image:51
```

Here the all of images under the project 51 would be transferred. It is not necessary to specify intermediate objects in the hierarchy and so:

```
$ omero chown 5 Project/Image:51
```

would have the same effect as the call above.

Transferring all objects belonging to specified users

Note that this feature is advanced and might be potentially slow. To transfer ownership of all objects belonging to a user or group of users the following form of the user specifier is used.

```
$ omero chown 10 Experimenter:1,3,7
```

Here ownership of all the objects belonging to users 1, 3 and 7 would be transferred to user 10.

Including and excluding objects

--include

Linked objects that would not ordinarily be transferred can be included in the transfer using the *--include* option:

```
$ omero chown 5 Image:51 --include Annotation
```

This call would move any annotation objects linked to the image.

--exclude

Linked objects that would ordinarily be transferred can be excluded from the transfer using the *--exclude* option:

```
$ omero chown 5 Project:51 --exclude Dataset
```

This will transfer project 51 but not any datasets contained in that project.

The two options can be used together:

```
$ omero chown 5 Project/Dataset:53 --exclude Image --include FileAnnotation
```

This will transfer any datasets under project 53, that are not otherwise contained elsewhere, excluding any images in those datasets but including any file annotations linked to the moved datasets. In this case the images that are not otherwise contained in datasets will be orphaned.

Further options

--ordered

Move the objects in the order specified.

Normally all of the specified objects are grouped into a single transfer command. However, each object can be transferred separately and in the order given. Thus:

```
$ omero chown 5 Dataset:53 Project:51 Dataset:54 --ordered
```

would be equivalent to making three separate calls:

```
$ omero chown 5 Dataset:53
$ omero chown 5 Project:51
$ omero chown 5 Dataset:54
```

--report

Provide a detailed report of what is transferred:

```
$ omero chown 5 Project:502 --report
```

--dry-run

Run the command and report success or failure but does not transfer the objects. This can be combined with the `omero chown --report` to provide a detailed confirmation of what would be transferred before running the move itself.

Note that changing ownership requires elevated privileges and can only be carried out by full administrators, restricted administrators with the correct privileges, or group owners.

See also:

OMERO.cli as an OMERO admin tool

System administrator documentation for the Command Line Interface. **This includes guidance for managing groups and users which can be done by restricted administrators with the correct privileges.**

Command Line Interface as an OMERO development tool

Developer documentation for the Command Line Interface

1.3 Additional resources

- [OMERO for scientists](#) introduces OMERO for new users, while the [feature pages](#) provide an overview of the platform features by type, including community developed apps and integrations which could help OMERO meet your research needs more fully.
- You can try out OMERO without committing to installing your own server by applying for an account on our [demo server](#).
- The [User guides](#) offer workflow-based assistance.
- The [OME YouTube channel](#) features tutorials and presentations.
- As OMERO is an open source project with developers and users in many countries, [connecting to the community](#) can provide you with a wealth of experience to draw on for help and advice.
- Additional [OMERO apps](#) add functionality to the OMERO.web or Command-Line clients.

1.3.1 Community support

The [Open Microscopy Environment](#) provides a number of resources for both our user and developer communities to assist in use and development of our software. Contributions through our mailing lists and forums are always welcome.

Web

The Open Microscopy Environment website is at <https://www.openmicroscopy.org>. Bio-Formats can be found at <https://www.openmicroscopy.org/bio-formats>.

Forums

The primary support channel is the [forum](#). The legacy [OME forum](#) and the list archives for [ome-devel](#) and [ome-users](#) contain historical support topics and remain available as read-only.

1.3.2 What's new for OMERO 5.6 for users

Updates and new features for OMERO 5.6 include:

- Decoupled OMERO.py and OMERO.web to allow more frequent releases.
- Filter Images by Map Annotations in OMERO.web.

See the [User guides](#) for information on how to incorporate these new features into your current workflows.

1.3.3 CHANGELOGS

1.3.4 Links to decoupled repositories

Starting from OMERO 5.5, the following repositories have been decoupled.

omero-build:

- omero-gateway-java [CHANGELOG.md](#)
- omero-blitz [CHANGELOG.md](#)
- omero-server [CHANGELOG.md](#)
- omero-renderer [CHANGELOG.md](#)
- omero-romio [CHANGELOG.md](#)
- omero-common [CHANGELOG.md](#)
- omero-model [CHANGELOG.md](#)

omero clients:

- omero-matlab [CHANGELOG.md](#)
- omero-insight [CHANGELOG.md](#)
- omero-py [CHANGELOG.md](#)
- omero-web [CHANGELOG.md](#)

1.3.5 OMERO version history

5.6.10 (December 2023)

This release includes the following upgrade of the OMERO.server Java components:

- omero-gateway-java 5.9.0
- omero-blitz 5.7.2
- omero-server 5.6.10
- omero-renderer 5.5.15
- omero-romio 5.7.5
- omero-common 5.6.4
- omero-model 5.6.13

Improvements include:

- an upgrade of logback-classic to version 1.3.14
- an upgrade of Bio-Formats to version 7.1.0

Note: This upgrade will invalidate the Bio-Formats Memoizer cache. Please see the upgrade guide for further information.

This version of the OMERO.server has been tested with:

- OMERO.py 5.17.0
- OMERO.web 5.23.0

5.6.9 (October 2023)

This release includes the following upgrade of the OMERO.server Java components:

- omero-gateway-java 5.8.2
- omero-blitz 5.7.1
- omero-server 5.6.9
- omero-renderer 5.5.14
- omero-romio 5.7.4
- omero-common 5.6.3
- omero-model 5.6.12

Improvements include:

- remove support for ADH and let Ice choose the default SSL protocol.
- revert “Java client init: no “anon” among jdk.tls.disabledAlgorithms”
- an upgrade of Bio-Formats to version 7.0.0

Note: This upgrade will invalidate the Bio-Formats Memoizer cache. Please see the upgrade guide for further information.

This version of the OMERO.server has been tested with:

- OMERO.py 5.16.0

- OMERO.web 5.22.1
- OMERO.dropbox 5.6.2

5.6.8 (July 2023)

This release includes the following upgrade of the OMERO.server Java components:

- omero-gateway-java 5.8.1
- omero-blitz 5.6.3
- omero-server 5.6.8
- omero-renderer 5.5.13
- omero-romio 5.7.3
- omero-common 5.6.2
- omero-model 5.6.11

Improvements include:

- support histogram for tiled images and fix generation of histogram for float images
- add default scanning period in logback.
- an upgrade of Bio-Formats to version 6.14.0

Note: This upgrade will invalidate the Bio-Formats Memoizer cache. Please see the upgrade guide for further information. We also recommend to re-index the database especially if the OMERO server has a large number of High Content Screening data.

This version of the OMERO.server has been tested with:

- OMERO.py 5.14.0
- OMERO.web 5.22.0
- OMERO.dropbox 5.6.2

5.6.7 (March 2023)

This release includes the following upgrade of the OMERO.server Java components:

- omero-gateway-java 5.8.0
- omero-blitz 5.6.2
- omero-server 5.6.7
- omero-renderer 5.5.12
- omero-romio 5.7.2
- omero-common 5.6.1
- omero-model 5.6.10

Improvements include:

- address performance issues when indexing fileset
- add `omero.search.max_fileset_size` property to indicated the maximum size of the fileset to be considered for indexing

- run the `PixelDataThread` Application events in `SYSTEM` Thread pool
- declare `logback-classic` as explicit dependency and set to `1.2.x`
- an upgrade of `Bio-Formats` to version `6.12.0`

Note: This upgrade will invalidate the `Bio-Formats` Memoizer cache. Please see the upgrade guide for further information. We also recommend to re-index the database especially if the `OMERO` server has a large number of High Content Screening data.

This version of the `OMERO.server` has been tested with:

- `OMERO.py` 5.13.1
- `OMERO.web` 5.19.0
- `OMERO.dropbox` 5.6.2

5.6.6 (December 2022)

This release includes the following upgrade of the `OMERO.server` Java components:

- `omero-gateway-java` 5.7.0
- `omero-blitz` 5.6.0
- `omero-server` 5.6.5
- `omero-renderer` 5.5.11
- `omero-romio` 5.7.1
- `omero-common` 5.6.0
- `omero-model` 5.6.9

as well as the upgrade of `omero-scripts` to version `5.7.1`.

Improvements include:

- enhancement of the Java Command line importer developer user experience
- the option for `omero.server.nodedescriptors` to be queried from a client
- a new property `omero.qa.feedback` to configure the QA system the feedback is submitted to.
- an upgrade of `Bio-Formats` to version `6.11.1`

Note: This upgrade will invalidate the `Bio-Formats` Memoizer cache. Please see the upgrade guide for further information.

This version of the `OMERO.server` has been tested with:

- `OMERO.py` 5.13.1
- `OMERO.web` 5.16.0
- `OMERO.dropbox` 5.6.2

5.6.5 (June 2022)

This release includes the following upgrade of the OMERO.server Java components:

- omero-gateway-java 5.6.10
- omero-blitz 5.5.12
- omero-server 5.6.4
- omero-renderer 5.5.10
- omero-romio 5.7.0
- omero-common 5.5.10
- omero-model 5.6.7

as well as the upgrade of omero-scripts to version 5.7.0.

Improvements include:

- a new server configuration allowing to control the pyramidal requirement for floating-point images
- an upgrade of Bio-Formats to version 6.10.0
- the inclusion of the OMEZarrReader version 0.3.0 for reading OME-NGFF data

Note: This upgrade will invalidate the Bio-Formats Memoizer cache. Please see the upgrade guide for further information.

This version of the OMERO.server has been tested with:

- OMERO.py 5.11.2
- OMERO.web 5.14.1
- OMERO.dropbox 5.6.2

5.6.4 (April 2022)

This release improves and expands integration testing to handle Django 3.2.x. It also removes obsolete code. More importantly, it has been tested with:

- omero-blitz 5.5.10
- omero-gateway-java 5.6.9
- omero-py 5.11.1
- omero-web 5.14.0
- omero-dropbox 5.6.2
- omero-scripts 5.6.2

Note: This upgrade will invalidate the Bio-Formats Memoizer cache. Please see the upgrade guide for further information.

5.6.3 (October 2020)

This release improves and expands integration testing and removes obsolete code. More importantly, it has been tested with:

- omero-blitz 5.5.8
- omero-gateway-java 5.6.5
- omero-py 5.8.1
- omero-web 5.8.1
- omero-dropbox 5.6.2

5.6.2 (July 2020)

This release adds installation documentation for server and web on CentOS 8 and Ubuntu 20.04. We have dropped support for Ubuntu 16.04 and removed the corresponding installation instructions.

This version has been tested with:

- omero-blitz 5.5.7
- omero-gateway-java 5.6.4
- omero-py 5.7.1
- omero-web 5.7.0
- omero-dropbox 5.6.2

Note: This upgrade will invalidate the Bio-Formats Memoizer cache. Please see the upgrade guide for further information.

5.6.1 (March 2020)

Security release focused on fixing [vulnerabilities](#) 2019-SV1 through 2019-SV6. This version has been tested with:

- omero-blitz 5.5.6
- omero-gateway-java 5.6.3
- omero-py 5.6.2
- omero-web 5.6.3
- omero-dropbox 5.6.1

Note: This upgrade will invalidate the Bio-Formats Memoizer cache. Please see the upgrade guide for further information.

5.6.0 (January 2020)

First release of OMERO with support for Python 3. All Python code has been removed from the distributed ZIP file and will need to be installed from PyPI. This version has been tested with:

- omero-blitz 5.5.5
- omero-gateway-java 5.6.2
- omero-py 5.6.0
- omero-web 5.6.1
- omero-dropbox 5.6.1

5.5.1 (July 2019)

Bug fix release focusing on installation issues that were seen with 5.5.0 as well as an upgrade of Bio-Formats to 6.1.1.

- **web:**
 - Allow the customization of the web logo
 - Allow overriding server configuration
 - Dynamically look up client download links
 - Fix description in new Project, Dataset etc.
 - Fix layout of the user account form
- **Java gateway:**
 - New methods added to allow change group of objects
 - New methods added to load objects (datasets, etc.) by name
 - New methods added to get original and repository paths of images
 - Minor fixes in createDataset and getPixelSize methods
- **import:**
 - Add import target support for creating Projects
- **scripts:**
 - Enable annotating Projects and Datasets with the Populate Metadata script
- **server:**
 - Fix SSL cipher issue to allow Insight to be used from Fedora 30
 - Fix issue with loading Hibernate's DTD when offline
 - Properly close OMERO.tables which kept sessions alive

Note: Due to the stricter closing of OMERO.tables, it may be necessary to update plugins like omero-metadata which previously were leaking files.

5.5.0 (June 2019)

This version **does not** require a database upgrade.

For more information about the aim of the 5.5 series and future plan, please read our [blog post](#).

This release focuses on dropping support for Java 7, Python 2.6 and Ice 3.5, adding support for Java 11 and PostgreSQL 10, and on decoupling the Java components to new, separate repositories, each with a new [Gradle](#) build system:

- <https://github.com/ome/omero-dsl-plugin>
- <https://github.com/ome/omero-model>
- <https://github.com/ome/omero-common>
- <https://github.com/ome/omero-romio>
- <https://github.com/ome/omero-renderer>
- <https://github.com/ome/omero-server>
- <https://github.com/ome/omero-blitz>
- <https://github.com/ome/omero-gateway-java>
- <https://github.com/ome/omero-blitz-plugin>
- <https://github.com/ome/omero-insight>
- <https://github.com/ome/omero-matlab>
- <https://github.com/ome/omero-javapackager-plugin>
- <https://github.com/ome/omero-api-plugin>

This has the goal of enabling more fine-grained releases.

A new restriction is that the names of server configuration properties may include only letters, numbers and the symbols “.”, “_”, “-”.

New plugins like `omero-artifact-plugin` allow reducing the boilerplate code in the build scripts of the decoupled repositories. Though initially disruptive, we hope this modernization and modularization will ease participation in the development of OMERO.

Additionally, this release improves the Web interface when OMERO is opened to the public and contains some useful CLI improvements.

- **build:**
 - Remove the generation of Ivy dependencies.html files
- **web:**
 - Introduce Advanced Search to allow and/or search options
 - Fix description in new Project, Dataset etc.
 - Allow user to create new Map Annotations for multiple selected objects at once
 - Fix date display
 - Remove / from 3rdparty in `ome.viewportImage.js`
 - Remove usage of deprecated calls
 - Remove apache config
 - Do not break display of Tag dialog when large font size is configured in browser

- Disable refresh button while existing refresh in progress
- Improve public user support
- Add ability to customize incorrect login text
- **cli:**
 - Disable foreground indexing
 - Improve logging of error when importing data via cli command
 - Clearly indicate empty log files when running a diagnostic
 - Fix bug when running *config load* passing a directory instead of a file
 - Add option to delete keys from map annotations
 - Add error code discovery
 - Deprecate the CLI upload module and plugin

5.4.10 (January 2019)

This release addresses a login issue for Java clients such as OMERO.insight. New releases of Java include a change to the `java.security` file that disables anonymous cipher suites. This change causes `SSLHandshakeException` when the client attempts to authenticate to OMERO.blitz. The OMERO 5.4.10 release has some clients check the security property `jdk.tls.disabledAlgorithms` for the value “anon” and remove it if present thus allowing authentication to proceed.

5.4.9 (October 2018)

This release addresses a critical import issue where files can be silently skipped.

Import improvements include:

- `ImportCandidates` returns filesets even when files are shared between several filesets independently of the scanning order
- insight: bug fixes for the lightweight importer UI

Other improvements include:

- BlitzGateway: new API to read `OriginalFile` as file-like objects
- server: add code to dispose of *Graphics* objects in the server
- Javadoc: add links to developer documentation for graph operations

5.4.8 (September 2018)

This release focuses on a number of import performance improvements while including several other fixes as well as an upgrade of Bio-Formats.

Import improvements include:

- cli: new experimental *–parallel-upload* and *–parallel-fileset* flags to the *import* command
- cli: new *fs importtime* cli command
- cli: add support for *–skip*, *–parallel-upload*, *–parallel-fileset* and *–readers* options in bulk import configuration files

- insight: new options for skipping various import steps to speed up the process (match cli's *--skip* option)
- insight: supporting imports with thousands of files by providing a lightweight UI
- insight: new loading placeholder when browsing data to show when an image is busy being processed and not ready to generate a thumbnail
- insight: added error placeholder when browsing data to indicate a failed import
- server: preventing recalculation of checksums for upload
- server: providing better performance logging, accessible to users via *fs logfile*
- as well as preservation of Bio-Formats' knowledge of channel colors where provided.

Other client changes include:

- web: better handling of large numbers of channels
- web: fixed socket leakage on unclosed web sessions
- web: fixed issue with bulk annotation table handling
- cli: deprecating *admin sessionlist* and *config list*

Sysadmin improvements include:

- new *%thread%* option for *omero.fs.repo.path* as well as fix a few bugs for dealing with parallel imports
- new *omero.threads.background_threads* property to limit the number of simultaneous imports

This release also upgrades the version of Bio-Formats which OMERO uses to 5.9.2.

5.4.7 (July 2018)

This is a security release which also includes a number of bug fixes. **It is highly recommended that you upgrade your server.**

See the [security advisories](#) page for details on 2018-SV1, 2018-SV2 and 2018-SV3.

Impacts of the security vulnerability fixes include:

- *omero.security.password_required=false* no longer applies for administrators: their correct password is always required
- administrators can no longer change the password of other administrators who are more privileged in any way
- administrators can no longer reset their password and receive the new one by e-mail: they must instead have another administrator who is at least as privileged set a new password manually
- cli: the session UUID has been removed from the standard output when logging in but can still be retrieved using *bin/omero sessions key*

Improvements include:

- web: fix loss of privileges when editing full admins
- web: fix exceptions on invalid connections
- web: fix CSS in group/user search element
- web: fix error when public user is disabled
- web: gray out user role when editing root user
- insight: permit *open_with* on original files
- read-only: reduce error logging for scripts and pixel data

- scripts: improve error messages for invalid MATLAB
- as well as various documentation improvements

Sysadmin improvements include:

- log locale and time zone information on startup

Developer updates include:

- cli: clean up “communicator not destroyed” logging
- cli: don’t hang when incorrect password passed in a script
- java: add a map annotation example
- java: throw a clear exception when -1 is used for all groups
- web: fix @render_response when extending base templates
- matlab: contributions from Kouichi Nakamura for working with annotations

This release also upgrades the version of Bio-Formats which OMERO uses to 5.9.0. **Note:** this is a significant upgrade and will invalidate the Bio-Formats Memoizer cache. Please see the upgrade guide for further information.

5.4.6 (May 2018)

This introduces a significant new subsystem for read-only operation with which servers can be configured not to make changes to the database, the filesystem, or both. The goal is to permit horizontal scaling of OMERO by running multiple servers in parallel to increase the throughput of data and metadata for large-scale analysis or publishing. Additionally, a read-only copy of an existing OMERO can be opened safely to the public for experimentation. For example, this infrastructure supports the public OMERO web and the Jupyter environment of the [Image Data Resource](#). Information on how to configure a read-only server is available at [Clustering](#).

Further improvements include:

- enabled big image support in ImageJ/Fiji
- reduced the number of threads used by OMERO.web
- fixed other bugs in OMERO.web including:
 - broken History tab
 - handling of script params
 - pagination calculations
 - public user login
 - browsing to user’s data in IE
- fixed the chosen login ports for OMERO.cli

Developer updates include:

- a new command to set custom physical pixel size using OMERO.cli
- deprecated Repository::pixels, TinyImportFixture and OMEROImportFixture
- improved test infrastructure
- reduced background events in the center panel plugin when not displaying Thumbnails
- added extra controls when specifying map and gamma in the rendering engine

This release also upgrades the version of Bio-Formats which OMERO uses to 5.8.2. **Note:** this is a significant upgrade and will invalidate the Bio-Formats Memoizer cache. Please see the upgrade guide for further information.

5.4.5 (March 2018)

This is a bug-fix release reactivating the thumbnail cache inadvertently disabled in 5.4.4 while fixing a pyramid issue.

Improvements include:

- reactivated thumbnail caching
- improved removepyramids help
- fixed display of thumbnails when searching for images by ID
- increased OMERO.web log size
- fixed CLI config list subcommand
- fixed leaking services in OMERO.py
- improved rendering of non-tile large images using OMERO.py and webgateway

This release does not upgrade the version of Bio-Formats which OMERO uses, which remains at 5.7.3.

5.4.4 (March 2018)

This is a bug-fix release which also introduces some new functionality.

It includes a security fix for [2017-SV6](#). **It is highly recommended that you upgrade your server.**

Improvements include:

- images can now be filtered by Tag in the center panel of OMERO.web
- enabled search by “File” and “Tag” annotations separately in OMERO.web, as opposed to only being able to search by All annotations
- fixed switching between grid display and thumbnail display in OMERO.web
- fixed the image preview and disabled projection in OMERO.insight when trying to project an image with all the channels turned off
- fixed parsing of polygons and polyline ROIs so they can be opened in ImageJ
- fixed creation of OMERO pyramids for little-endian files
- improved error message when login fails for OMERO.insight
- improved handling of idle connections in OMERO.insight
- improved loading speed of LUT
- OMERO.insight and OMERO.importer are now compatible with Java 9

Sysadmin improvements include:

- improved installation documentation for OMERO.web, and OMERO.server on Debian 9, Ubuntu 16.04 and CentOS 7
- added an admin command and script to allow deletion of corrupted pyramids created by a bug introduced with OMERO 5.2 (new uncorrupted pyramids can then be generated - see [OMERO.server upgrade](#) for details)
- allowed enforcement of a secure connection when importing data
- added commands to the CLI sessions plugin to enable the creation and removal of user sessions

Developer updates include:

- improved test infrastructure and coverage
- allowed filtering by namespace (ns) in webclient, API and annotations
- added support for more rendering parameters to the API
- added the option to respect a specific tile size
- added a method to load planes using JavaGateway
- added an example to the documentation for using “sudo” to create sessions for others with the JavaGateway
- documentation is now compatible with Sphinx 1.7

This release does not upgrade the version of Bio-Formats which OMERO uses, which remains at 5.7.3.

5.4.3 (January 2018)

This is a bug fix release for a resource leak in `omero.gateway.BlitzGateway` introduced with 5.4.2 that caused long-running processes to hang. No other changes are included.

5.4.2 (January 2018)

This is a bug-fix release.

Improvements include:

- added documentation on a complete workflow for publishing data from OMERO.server
- added references to the new OMERO pyramid format documentation (within the OME Data Model and File Formats documentation)
- faster loading of thumbnails for large Plates after a recent regression
- made projecting images belonging to another user only possible for users with the required permissions to save the new images
- improved the public user experience for password-less access
- updated SwingX library version used by OMERO.insight to stop insight-ij plugin crashing in Fiji
- CLI updates:
 - `import --target` into a container without the necessary permissions now fails before file upload starts and more transparently
 - `admin mail` timeout is now configurable via `--wait`
 - added `admin log` command for inserting statements to the server log

Sysadmin changes include:

- added warning about the need to regenerate your NGINX config for every upgrade
- fixed documentation bug affecting OMERO-version-specific guidance
- improved OMERO.tables startup stability
- server performance improvements and reduction in ERROR logging

Developer updates include:

- extended Python and Java examples to include Map Annotations and histograms

- added methods for updating OMERO.tables
- Java Gateway fixes for sessions and rendering
- fixed retrieval of Plate thumbnail URLs
- improved ‘Editing OMERO.web’ documentation
- improved Slice documentation for API deprecations
- added instructions to [Extensions](#) on how to create CLI plugins that are `pip` installable
- substantial effort to make third-party repositories easily testable; see [omero-test-infra](#) for more information

This release also upgrades the version of Bio-Formats that OMERO uses to 5.7.3.

5.4.1 (November 2017)

This is a bug-fix release.

Improvements include:

- labeled zoom slider bars in the UI to differentiate from horizontal scrollbars and make clear thumbnails can be zoomed (including Plate and Well thumbnails)
- fixes for installation walkthrough documentation - installation of script dependencies and gunicorn, and clarification of which user account to use for `pip install` actions
- fixed checking of “guest” user
- update to fetch third-party artifacts over https to allow OMERO to build even without a local Maven cache already populated
- added `javafx.activation` dependency to allow OMERO.insight to work with Java 9
- import of files reporting extreme pixel sizes now fail rather than hanging
- pyramid-making now aborts when a tile fails
- various test fixes
- CLI fixes:
 - improved help output for graphs commands to make it clearer that `--include` and `--exclude` expect class names not object IDs
 - allowed setting the `OMERO_PASSWORD` environment variable instead of using the `-w` command-line option
 - made passwords hidden by default when running `omero config get`
 - fixed the CLI metadata tabletest plugin to not use an empty list of Columns

This release also upgrades the version of Bio-Formats that OMERO uses to 5.7.2.

5.4.0 (October 2017)

A full, production-ready release of OMERO 5.4.0; featuring a new configurable user role “Restricted Administrators”; further improvements to OMERO.web; additions to OMERO.cli; and many fixes and performance improvements:

- added *Administrators with restricted privileges* to allow sysadmins to delegate management tasks to facility managers without granting them full system admin privileges, or to allow trusted users such as image analysts to carry out tasks on behalf of all other users
- fixed color conversion to RGBA
- added support for exporting images in a plate as OME-TIFF
- improved creation of rendering settings for images without any stats e.g. 32bit images
- improved performance for moving large Plates
- fixed projection of images if the range of timepoints specified is not the full range
- added support for transferring ownership of all the data of a given user using CLI
- renamed “Reverse Intensity” command to “Invert” in image viewers
- added support for ImageColumn with Screen and Plate targets in the populate_metadata script
- OMERO.web UI fixes:
 - improved display of Plates and Wells
 - fixed label position for Wells
 - added the ability to display Image and Well metadata in the Tables section for the same Plate
 - improved copy/paste of rendering settings workflow
 - improved layout of left-hand panel including the position of the search panel
 - added support for administrators with restricted privileges to create Project/Dataset for other users
 - rolled back the display of tables in the viewer
 - fixed forgotten password functionality

Sysadmin changes include:

- added support for the creation of administrators with restricted privileges in OMERO.web admin panel
- added method to create administrators with restricted privileges specifying a password
- added specific installation instructions for Debian 9
- added configuration to limit queries that public users can do in OMERO.web
- created minimal NGINX configuration file that can be included in a fixed file to allow custom NGINX options to be defined only once (e.g. SSL options)
- installed django-redis by default
- CLI improvements and fixes:
 - fixed admin plugin so “cleanse” can handle larger directories
 - added to chown plugin ability to target all of given users’ data
 - adjusted handling of standard input
 - added infrastructure to load external CLI plugins
 - dropped support for command `admin ports`

Developer updates include:

- added method to JavaGateway to manipulate admin privileges
- fixed issue with JSONP decorator
- removed SciPy dependency
- adjusted OMERO.blitz API to allow some query results to be cached
- added support to the rendering engine to update a series of settings in one call
- added method to OMERO.py to manipulate advanced rendering settings
- allowed the Maven repository to be overridden
- removed unused 3rd party libraries in OMERO.web
- added support for PyTables version 3.4+
- deprecated Path Object in OMERO Model
- updated the documentation for server installation on Mac OS to no longer use the homebrew formulae from <https://github.com/ome/homebrew-alt> (these are not working and will not be fixed)

Further changes to the Python BlitzGateway are described in *What's new for OMERO 5.6 for developers*.

This release also upgrades the version of Bio-Formats which OMERO uses to 5.7.1.

5.3.5 (October 2017)

This is a security release - see the [security advisory](#) for further details.

It is highly recommended that you upgrade your server.

5.3.4 (September 2017)

This is a security release - see the [security advisory](#) for further details.

This release also upgrades the version of Bio-Formats which OMERO uses to 5.5.3.

It is highly recommended that you upgrade your server.

5.3.3 (June 2017)

This is a bug-fix release.

Improvements include:

- support for two new lookup tables from [Janelia](#)
- fixed loading of Well in right-panel when browsing Well under Tag tree or from search results
- fixed rotation of labels in figure scripts

Sysadmin changes include:

- clarified the upgrade of the “Open With” option
- allowed installation of OMERO.web with ice 3.5
- fixed recursive loading of feedback in OMERO.web

- provided patch for OMERO.server installation on OS using OpenSSL 1.1.0 e.g. Debian 9 see [Troubleshooting OMERO](#)

Developer updates include:

- added an example of how to retrieve shapes from a ROI using batch querying for scalability
- improved logging of errors during deletion
- added new methods to Java Gateway
- improved login options in Java Gateway
- specified an image's dataset in its URL to give more context to OMERO.web apps

This release also upgrades the version of Bio-Formats which OMERO uses to [5.5.2](#).

5.3.2 (May 2017)

This is a bug-fix release.

Improvements include:

- improved populate_metadata plugin
- fixed deletion of a range of objects from CLI
- textual annotations without a namespace can now be added at import using the CLI
- improved thumbnails retrieval in OMERO.web
- added “Open With” option to the right-hand panel in OMERO.web
- private group owners are now not offered the ability to annotate other people's data in OMERO.web UI, an action which was not permitted by the server anyway
- preview of wells now available in the right-hand panel

Sysadmin changes include:

- made the Django middleware classes configurable using a new property
- added property to allow connections from specified origins (CORS)
- administrators can now use the CLI to move data between groups without belonging to those groups
- for OMERO.web apps to be available via “Open With” option, administrators need to use the “omero.web.open_with” configuration option

Developer updates include:

- exposed more enumerations from ome-model
- added ROIs support to the Web API

This release also upgrades the version of Bio-Formats which OMERO uses to [5.5.0](#).

5.3.1 (April 2017)

This is a bug-fix release focusing on shares.

Improvements include:

- enabled viewing images in share
- enabled importing hidden image files (Windows client issue)
- clarified installation of OMERO.web
- saved polygon and polyline as defined in the OME model
- fixed viewing of images without pixels size
- added support for large image export as jpeg/png from OMERO.insight

This release also upgrades the version of Bio-Formats which OMERO uses to 5.4.1.

5.3.0 (March 2017)

A full, production-ready release of OMERO 5.3.0; featuring a major reworking of OMERO.web and web apps; dropping support for Windows for the server and for deploying OMERO.web using Apache; and introducing new user features and many fixes and performance improvements:

- improved support for many file formats via Bio-Formats 5.4.0
- introduced ROI Folders
- new UI for displaying Screen Plate Well data in OMERO.web and OMERO.insight
- support for lookup tables and reverse intensity rendering
- color mapping for multiple channels without set colors has been improved to use RGBRGB rather than RGBBB (i.e. to loop through red, green, blue rather than setting all later channels to blue)
- support for histograms in the clients and server
- ability to filter by ratings in OMERO.web
- added 'Open With...' functionality to OMERO.web
- color of shapes is now handled according to the data model, using RGBA rather than ARGB format (an sql script is available to upgrade existing shapes; this will not happen automatically as part of the OMERO upgrade)
- improved performance for moving and deleting data
- Wells can now be annotated and searched by annotations
- enabled downloading/exporting of plate data
- improved reading of tables data
- script improvements including ability to create tiled images from big ROIs, fixes for creating standard images from ROIs, and to stop the Combine_Images script from ignoring pixel sizes set on the target images
- names for plates and images set in the metadata read by Bio-Formats are now imported into OMERO and the filename (which was previously used) is only used where an alternative has not been set
- many bug fixes

Sysadmin changes include:

- added support for Ice 3.6.3
- official OMERO.web apps are now all installable from PyPI

- OMERO.web has been decoupled from the server and can now be deployed separately
- dropped support for Windows for OMERO.server
- OMERO.web deployment via Apache is no longer supported
- OMERO.web also now requires Python 2.7
- CLI improvements including updates to the import output to make it more usable by scripts etc.
- options added for customizing the tree in OMERO.web
- introduced hide-password option in CLI
- new options added to `omero config`
- removed deprecated client menu properties

Developer updates include:

- performed major code cleanup
- major Web API rework
- adjustment to support the upcoming Java 1.9
- made python testing package public so it can be used by external clients
- improved build system integration with local Maven
- made Scripts repository and official OMERO.web apps pep8 and flake8 compatible
- removed restriction on name length
- added support for enumeration changes
- utils script classes deprecated
- deprecated shares
- deprecated search bridges
- disabled jquery cache

Further details on breaking changes are available on [What's new for OMERO 5.3 for developers](#). Work on the Web API is ongoing and will include moving away from the use of JSONP and introducing Django CORS.

5.2.8 (March 2017)

This is a security release including three security vulnerability fixes.

<https://www.openmicroscopy.org/security/advisories/2017-SV1-filename> prevents users from accessing and manipulating other people's data by creating an original file and changing its path to point to another user's file on the underlying filesystem.

<https://www.openmicroscopy.org/security/advisories/2017-SV2-edit-rw> prevents users in read-write groups from editing official scripts.

<https://www.openmicroscopy.org/security/advisories/2017-SV3-delete-script> prevents the deletion of official scripts by users without the correct permissions to do so.

It is highly recommended that you upgrade your server.

5.2.7 (December 2016)

This is a release aimed at system administrators or developers wanting to build OMERO with Ice 3.6.3.

This release also upgrades the version of Bio-Formats which OMERO uses to 5.1.10.

All scripts handling Regions of Interest (ROIs) now support ROI not linked to any plane as defined by the OME Model.

5.2.6 (October 2016)

This is a bug-fix release focusing on services closure and a DB upgrade fix. Improvements include:

- fixed closure of session in Java when using Ice 3.5
- fixed memory leak where services were not correctly closed
- added a DB upgrade patch to fix errors only affecting DBs that have been upgraded from OMERO 4.4
- fixed a MATLAB regression introduced in 5.2.2, casting error.
- fixed error in logs on getProjectedThumbnail

Support for OMERO.web deployment using Apache has also been deprecated and is likely to be removed during the 5.3.x line.

5.2.5 (August 2016)

This is a security release to fix the access privileges of the share function, which were potentially allowing users to access private data belonging to other users via the API.

See <https://www.openmicroscopy.org/security/advisories/2016-SV2-share> for details. Shares will now respect user privileges as set by the group permission level. Note that Shares now **only** support images even when used via the API.

It is highly recommended that you upgrade your server. For those not in a position to do so as a matter of urgency, a workaround is provided which deletes all shares and disables their creation.

5.2.4 (May 2016)

This is a security release to fix the cleanse.py script used by the “bin/omero admin cleanse” command, which was not properly respecting user permissions and may lead to data loss.

See <https://www.openmicroscopy.org/security/advisories/2016-SV1-cleanse> for details. The script and command have now been made admin-only.

It is highly suggested that you upgrade your server or apply the patch available from the security page.

5.2.3 (May 2016)

A bug-fix release. Improvements include:

- fixed problem with float images
- all scripts currently exposed to users via our website have been reviewed and fixed where necessary so they are all now 5.2.x compatible, and a new omero-install workflow has been developed to ensure these are reviewed regularly going forward
- better support for metadata annotations in clients including tag/tagset support and performance issues
- fixes in OMERO.web for deleting MIFs

- improvements to the navigation of large datasets and display of plates in OMERO.web
- other OMERO.web bug fixes
- OMERO.insight and CLI import improvements
- other OMERO.insight bug fixes, including for downloading data

Developer updates include:

- Java gateway improvements

System administrator updates include:

- Ice 3.6.2 support for UNIX-like systems, including specific installation walkthroughs
- redis support for web sessions caching
- a fix to allow OMERO.web to be run in a Docker container
- improved OMERO.web configuration
- warnings added regarding the [end of Windows support in the 5.3.0 release](#) (note that we will be preparing a guide for migrating from Windows for existing servers and adding it to the documentation as soon as we can)

This release also upgrades the version of Bio-Formats which OMERO uses to 5.1.9.

5.2.2 (February 2016)

A bug-fix release which also introduces some new client features. Improvements include:

- display of ROI masks in OMERO.web image viewer
- display of OMERO.tables data for Wells in the OMERO.web right hand panel
- ‘Populate Metadata’ script to enable generation of OMERO.tables for Wells is now usable from both OMERO.web and OMERO.insight (note this is still in development and has some limitations)
- measurement tool fixes
- fixed pixel size metadata and scalebar in OMERO.web image viewer for images with pixel size units other than micrometer
- fixed OMERO.web handling of turning off interpolation of pixels
- previous and next buttons fixed in OMERO.web image viewer
- delete and change group performance improvements
- better handling of dates in search
- client support for map annotations in OME-TIFF
- disabled orphaned container feature
- OMERO.web clean-up to remove obsolete volume viewer

Developer updates include:

- Python API examples for creating Polygon and Mask shapes
- Python API example for “Populate Metadata” to create OMERO.tables for Wells
- OMERO.tables documentation extended
- updated ‘What’s New for developers’ to clarify that `pojos` has been renamed as `omero.gateway.model`
- dynamic scripts functionality documented

- dynamic loading of omero.client server settings into HTTP sessions

System administrator updates include:

- clarification of OMERO.web documentation for nginx deployment, including an experimental solution to resolve download issues
- documentation of hard-linking issues for in-place import on linux systems

Note that the OMERO Virtual Appliance has been discontinued and will not be updated for version 5.2.2 or any later releases.

This release also upgrades the version of Bio-Formats which OMERO uses to [5.1.8](#).

5.2.1 (December 2015)

A bug-fix release focusing on improving installation documentation and workflows. Other improvements include:

- bug fix for missing hierarchy when viewing High Content Screening data
- improvements to the right-hand panel in OMERO.insight
- measurement tool fixes
- OMERO.web fix for displaying size units

System administrator updates include:

- improved installation documentation, including detailed walkthroughs for specific OS
- OMERO.web deployment fixes

Developer updates include:

- OMERO Javadocs now link to the relevant version of Bio-Formats Javadocs for inherited methods
- clean-up of server dependencies
- jstree clean-up
- CLI graph operation improvements for deleting
- minimal-omero-client and pom-omero-client clean-up

This release also upgrades the version of Bio-Formats which OMERO uses to [5.1.7](#).

5.2.0 (November 2015)

A full, production-ready release of OMERO 5.2.0; dropping support for Java 1.6; featuring major upgrading of OMERO.web; re-working of the Java Gateway; and introducing new user features and many fixes and performance improvements:

- improved support for many file formats via Bio-Formats 5.1.5
- faster import for images with a large number of ROIs
- performance improvements for OMERO.web including faster data tree loading
- Java Web Start has been dropped, it is no longer possible to launch the desktop clients from the web
- reworked display of metadata and annotations in both UI clients
- many bugs fixed

Developer and system administrator updates include:

- the OMERO web framework no longer bundles a copy of the Django package, this dependency must be installed manually
- updated jstree to 3.08 and now using json for all tree loading to substantially improve performance
- removed FastCGI support, OMERO.web can be deployed using WSGI
- configuration property `omero.graphs.wrap` which allowed switching back to the old server code for moving and deleting data has now been removed. You should migrate to using the new graph request operations before 5.3 when the old request operations will be removed
- introduced new Java Gateway to facilitate the development of Java applications
- aligned OMERO Rect with OME-XML schema for ROI. Clients using the OMERO.blitz server API to work with ROIs will need to be updated

5.1.4 (September 2015)

A bug-fix release covering all components. Improvements include:

- channel buttons fixed in OMERO.web
- improved UI experience when moving annotated data between groups in OMERO.web
- improved performance for loading annotations in the right-hand panel of OMERO.web
- much better handling of ROIs covering large planes in OMERO.insight
- rendering setting fixes for copy and paste actions in OMERO.insight
- rendering fixes for floating point data
- Admins can now configure whether the clients interpolate images by default
- better formatting of Delta-T and exposure times in the clients
- directories are now preserved when downloading multiple original files
- various improvements to the OMERO-ImageJ handling of ROIs and measurements, including the ability to name measurement tables
- current session key can now be returned via the CLI
- other CLI improvements including usability of ‘chmod’ for downgrading group permissions, and listing empty tagsets
- added support for groups in OMERO.matlab methods

Developer updates include:

- improvements to web logging to log full request and status code
- fixed joda-time version mismatch
- cleanup of old insight code to remove remaining references to OMERO.editor

Support for deployment of OMERO.web using FastCGI has also been deprecated in this release and is scheduled to be removed in 5.2.0. Sysadmins should move to using WSGI instead. We are also intending to stop distributing Java Webstart for launching OMERO.insight from your browser, as security concerns mean browsers are increasingly moving away from supporting this type of application. You can read further information regarding this decision on our [Web Start blog post](#).

5.1.3 (July 2015)

A bug-fix release which also introduces some new functionality. Improvements include:

- tagging actions extended; you can now use tag sets to tag images on import
- tagging ome-tiff images at import has also been fixed
- greatly improved workflow and bug fixes for the Share functionality in OMERO.web which enables you to share images with users outside of your group (including removal of part of the UI)
- group admins and owners can now change ownership of data via the CLI
- better reporting for the ‘delete’ and ‘chgrp’ functionality in the CLI
- fixed display of images in plates with multiple acquisitions
- fixed export of results as .xls files from OMERO.insight
- improved workflow for ImageJ and OMERO interactions
- support for WSGI OMERO.web deployment
- fixed OMERO.mail service for web errors
- fixes for ROI display in OMERO.web (thanks to Luca Lianas of CRS4)
- fixes and workflow improvements for running scripts and script dialogs

Developer updates include:

- OMERO.web clean-up (removal of ‘-locked’)
- reorganization of the server bundle to move various licenses and dependencies under a new ‘share’ folder
- refactoring of ‘Chown2’, ‘Chmod2’, ‘Chgrp2’ and ‘Delete2’
- addition of dynamic scripts
- the ‘rstring’ implementation is now more lenient and should better handle unicode
- Bio-Formats submodule removed from OMERO; decoupling effort means OMERO now consumes the Bio-Formats release build from the artifactory

This release also includes the fix for the Java security issue, as discussed in the [recent blog post](#). Testing suggests this fix should not have any performance implications. You should upgrade your Java version to take advantage of the security fix.

5.1.2 (May 2015)

A bug-fix release which also introduces some new functionality. Improvements include:

- support for Read-Write groups
- the LDAP plugin can now set users as group owners whether on creation or via the improved sync_on_login option
- users logged into the webclient can now automatically log in via webstart
- results tables from ImageJ/Fiji can be attached to images in OMERO and the ImageJ/Fiji workflow has been improved
- better delete functionality and warnings in the UI
- improved graph operations like ‘delete’ and ‘chgrp’, as well as the new ‘chmod’ operation (for changing group permissions), are now used across the clients including the CLI

- an API for setting and querying session timeouts is now available via the CLI
- magnification now reflects microscopy values (e.g. 40x) rather than a percentage in both clients
- more readable truncation of file names in the OMERO.insight data tree
- OMERO.web fixes and improvements including:
 - interpolation
 - optimization of plate grid and right-hand panel
 - option to download single original files
 - significant speed-up in loading large datasets
- deployment fixes include:
 - new default permissions on the var/ directory
 - better checks of the DropBox directory permissions
 - new and some deprecated environment variables
 - a startup check for lock files on NFS
 - use /var/run for omero.fcgi

Critical bugs which were fixed include:

- the in-place import file handle leak (which was a regression in 5.1.1)
- various unicode and unit failures were corrected

5.1.1 (April 2015)

A bug-fix release focusing on user-facing issues and cleaning resources for developers. Improvements include:

For OMERO.web:

- significant review of the web share functionality
- correction of thumbnail refreshing
- fixes to the user administration panel
- fix for embedding of the Javascript image viewer

For OMERO.insight:

- improved open actions
- tidying of the menu structure
- correction of the mouse zoom behavior
- fix for the Drag-n-Drop functionality

Other updates include:

- overhaul of the CLI session log-in logic
- cleaning and testing of all code examples
- further removal of the use of deprecated methods

5.1.0 (April 2015)

A full, production-ready release of OMERO 5.1.0; updating the Data Model to the January 2015 schema, including support for units and new more flexible user-added metadata; and introducing new user features, new supported formats and many fixes and performance improvements:

- support for units throughout the Data Model allowing for example, pixel sizes for electron microscopy to be stored in nanometers rather than being set as micrometers
- new, searchable key-value pairs annotations for adding experimental metadata (replacing OMERO.editor, which has been removed)
- improved workflow for rendering settings in the UI and parity between the clients
- import images to OMERO from ImageJ and save ROIs and overlays from ImageJ to OMERO
- importing as another user, previously only available for administrators, is now usable by group owners as well, allowing you to import data that will then be owned by the user you import it for
- improved performance for moving and deleting data
- removed the auto-levels calculation for initial rendering settings to substantially speed up performance, by using the min/max pixel intensities, or defaulting to full pixel range where min/max is unavailable
- import times are much improved for large datasets such as HCS and SPIM data
- improved performance for many file formats and new supported formats via Bio-Formats (now over 140)
- new OMERO.mail feature lets admins configure the server to email users
- support for configuring the server download policy to control access to original file download for public-facing OMERO.web deployments
- many developer updates such as removal of deprecated methods, and updates to OMERO.web and the C++ implementation (see the 5.1.0-m1 to 5.1.0-m5 developer preview release details below and the ‘What’s New’ for developers page)

5.1.0-m5 (March 2015)

Developer preview release - **only intended as a developer preview for updating code before the full public release of 5.1.0. Use at your own risk.**

Changes include:

- implementation of OMERO.mail for emailing users via the server
- performance improvements for importing large datasets
- support for limiting the download of original files
- various fixes for searching and filtering map annotations and converting between units
- deprecation of IUpdate.deleteObject API method
- versioning of all JavaScript files to fix browser refresh problems
- clarifying usage of OMERO.web views and templates including RequestContext

5.1.0-m4 (February 2015)

Developer preview release - **only intended as a developer preview for updating code before the full public release of 5.1.0. Use at your own risk.**

Changes include:

- final Database changes - image.series is now exposed in Hibernate
- improved deletion performance
- client bundle clean-up
- other clean-up work including pep8 and removal of deprecated methods and components
- new Map annotations are now included in the UI and search functionality
- ImageJ plugin updates which allow
 - importing of images and saving ROIs to OMERO from within the plugin
 - viewing images stored in OMERO and their ROIs generated within OMERO from within the plugin
 - updating ROIs on OMERO-stored images within the plugin and saving these back to OMERO without needing to re-import the image
- OMERO.matlab updates re: annotations
- OMERO.tables internal HDF5 format has changed

With thanks to Paul Van Schayck and Luca Lianas for their contributions.

5.0.8 (February 2015)

This is a bug-fix release for one specific issue causing OMERO.insight to crash when trying to open the Projection tab for an image with multiple z-stacks.

5.0.7 (February 2015)

This is a bug-fix release covering a number of issues:

- rendering improvements including 32-bit and float support
- vast improvements in Mac launching (separate clients for your Java version)
- faster import of complex plates
- OMERO.dropbox improvements
- ROI and measurement tool fixes
- OMERO.matlab updates

5.1.0-m3 (December 2014)

Developer preview release - 3 of 4 development milestones being released in the lead up to 5.1.0. **Only intended as a developer preview for updating code before the full public release of 5.1.0. Use at your own risk.**

Changes affecting developers include:

- implementation of units in the OMERO clients
- conversions between units
- OMERO.web updates
- server-side Graph work to improve speed for moving and deleting
- OMERO.insight bug-fixes especially for ROIs

5.1.0-m2 (November 2014)

Developer preview release - 2 of 3 development milestones being released in the lead up to 5.1.0. **Only intended as a developer preview for updating code before the full public release of 5.1.0. Use at your own risk.**

Model changes include:

- units support, meaning units now have real enums
- minor fixes for model changes introduced in m1

The units changes mean that the following fields have changed:

- Plane.PositionX, Y, Z; Plane.DeltaT; Plane.ExposureTime
- Shape.StrokeWidth; Shape.FontSize
- DetectorSettings.Voltage; DetectorSettings.ReadOutRate
- ImagingEnvironment.Temperature; ImagingEnvironment.AirPressure
- LightSourceSettings.Wavelength
- Plate.WellOriginX, Y
- Objective.WorkingDistance
- Pixels.PhysicalSizeX, Y, Z; Pixels.TimeIncrement
- StageLabel.X, Y, Z
- LightSource.Power
- Detector.Voltage
- WellSample.PositionX, Y
- Channel.EmissionWavelength; Channel.PinholeSize; Channel.ExcitationWavelength
- TransmittanceRange.CutOutTolerance; TransmittanceRange.CutInTolerance; TransmittanceRange.CutOut; TransmittanceRange.CutIn
- Laser.RepetitionRate; Laser.Wavelength

Other changes that may affect developers include:

- ongoing C++ implementation improvements
- ongoing work to add unit support in OMERO.insight
- further flake8 work

- removal of webtest app from OMERO.web to a separate repository
- removal of deprecated methods in IContainer and RenderingEngine
- removal of deprecated services IDelete and Gateway
- Blitz gateway fixes
- CLI fixes
- ROI and tables work

5.0.6 (November 2014)

This is a critical security fix for two vulnerabilities:

- <https://www.openmicroscopy.org/security/advisories/2014-SV3-csrf>
- <https://www.openmicroscopy.org/security/advisories/2014-SV4-poodle>

It is strongly suggested that you upgrade your server and follow the steps outlined on the security vulnerability pages. Additionally, a couple of bug fixes for system administrators are included in this release.

5.1.0-m1 (October 2014)

Developer preview release - 1 of 3 development milestones being released in the lead up to 5.1.0. **Only intended as a developer preview for updating code before the full public release of 5.1.0. Use at your own risk.**

Model changes include:

- channel value has changed from an int to a float
- acquisitionDate on Image is now optional
- Pixels and WellSample types are no longer annotatable
- the following types are now annotatable: Detector, Dichroic, Filter, Instrument, LightSource, Objective, Shape
- introduction of a “Map” type which permits storing key-value pairs, and a Map annotation type which allows linking a Map on any annotatable object

Other changes that may affect developers include:

- strict flake8’ing of all Python code
- C++ build is now based on CMake and is hopefully much more user-friendly
- new APIs: SendEmail and DiskUsage
- the password table now has a “changed” field

5.0.5 / 4.4.12 (September 2014)

This is a critical security fix for two vulnerabilities:

- <https://www.openmicroscopy.org/security/advisories/2014-SV1-unicode-passwords>
- <https://www.openmicroscopy.org/security/advisories/2014-SV2-empty-passwords>

It is highly suggested that you upgrade your server and follow the steps outlined on the security vulnerability pages.

5.0.4 (September 2014)

This is a bug-fix release for the Java 8 issues. It also features a fix for uploading masks in OMERO.matlab.

You need to upgrade your OMERO server if you want to take advantage of further improvements in Bio-Formats support for ND2 files.

5.0.3 (August 2014)

This is a bug-fix release addressing a number of issues including:

- improved metadata saving in MATLAB
- many bug fixes for ND2 files
- several other bug fixes to formats including LZW, CZI, ScanR, DICOM, InCell 6000
- support for NDPI and Zeiss LSM files larger than 4GB
- export of RGB images in ImageJ
- search improvements
- group owner enhancements
- Webclient updates including multi-file download

To take advantage of improvements in Bio-Formats support for ND2 files, you need to upgrade your OMERO.server as well as your clients.

5.0.2 (May 2014)

This is a bug-fix release addressing a number of issues across all components, including:

- import improvements for large image datasets
- shared rendering settings
- better tagging workflows
- disk space usage reporting for OMERO.web admins
- OMERO.matlab annotation handling
- custom Web Start intro page templates
- searching by image ID

To take advantage of improvements in Bio-Formats support for .czi files, you need to upgrade your OMERO.server as well as your clients.

4.4.11 (April 2014)

This is a bug-fix release for the Java Web Start issue. You only need to upgrade if this is a blocker for you and you cannot upgrade to 5.0.x as yet. Also note that the OMERO.insight-ij plugin version 4.4.x no longer works for Fiji, we are working on a fix for this. Plugin version 5.0.x is unaffected.

5.0.1 (April 2014)

This is a bug-fix release addressing a number of issues across all components, including:

- code signing to fix the Java Web Start issues
- stability improvements to search
- MATLAB fixes
- improvements to groups, user menus, file name settings etc
- new import scenario documentation covering ‘in-place’ importing.

5.0.0 (February 2014)

This represents a major change in how the OMERO server handles files at import compared with all previous versions of OMERO. Referred to as ‘OMERO.fs’, this change means that OMERO uses Bio-Formats to read your files directly from the filesystem in their original format, rather than converting them and duplicating the pixel data for storage. In addition, it continues our effort to support new multidimensional images. The changes are especially important for sites working with large multi-GB datasets, e.g. long time lapse, HCS and digital pathology data.

4.4.10 (January 2014)

This is a bug-fix release addressing a number of issues across all components, including:

- improved tile-loading
- better network-disconnect handling
- more flexible
- webapp deployment
- Ice 3.5.1 support (except Windows)
- improved modification of metadata, users and groups

4.4.9 (October 2013)

This is a bug-fix release addressing a number of issues across all components, also including:

- Ice compatibility issues
- new scripting sharing service
- new user help website
- new partner project pages.

The minimum system requirement is Java 1.6 (Java 1.5 is no longer supported).

A security vulnerability was identified and resolved, meaning that we strongly recommend all users upgrade their OMERO clients and servers.

4.4.8p1 (July 2013)

This is a patch release addressing a network connection problem in the clients introduced by a new version of Java.

4.4.8 (May 2013)

This is bug-fix release addressing two specific issues: a problem with the OMERO.insight client for Linux, and image thumbnails not loading for Screens/Plates in Private/Read-Only groups in OMERO.web. You only need to upgrade if you are an OMERO.insight user on Linux or you are using OMERO.web to view HCS data in Private or Read-Only groups.

4.4.7 (April 2013)

This is a point release including several new features and fixes across all components. This includes improvements in viewing of ‘Big’ tiled images, new permission features, new OMERO.web features, and several utility functions in OMERO.matlab.

4.4.6 (February 2013)

This is bug-fix release addressing a number of issues across all components. This includes a major fix to repair the C++ binding support for Ice 3.4. There has also been a potentially breaking update to the CLI.

4.4.5 (November 2012)

This is bug-fix release focusing on improvements to the OMERO clients. OMERO.web now supports “batch de-annotation”, filtering of images by name and improved export to OME-TIFF and JPEG. OMERO.insight has fixes to thumbnail selection and image importing and exporting.

4.4.4 (September 2012)

This is a bug-fix release addressing a number of issues across all components.

- OMERO.insight fixes include connection and configuration options and tagging on import.
- OMERO.web improvements include big image and ROI viewer fixes, improved admin and group functionality and rendering/zooming fixes.
- OMERO.server now has improved LDAP support and VM and homebrew deployments as well as fixes for file downloads above 2GB, permissions, memory leaks and JDK5.

4.4.3 (August 2012)

This is a critical security fix for:

- <https://www.openmicroscopy.org/security/advisories/2012-SV1-ldap-authentication>

Anyone using OMERO 4.4.2 or earlier with LDAP authentication should immediately upgrade to 4.4.3.

4.4.2 (August 2012)

This release is a major bug fix for archiving files larger than 2 GB. If you do not archive files larger than 2 GB, you do not need to upgrade your clients or your server. There is also a minor fix for an OMERO.imagej plugin security issue, but it is only necessary to update the version of Bio-Formats that is installed in ImageJ.

4.4.1 (July 2012)

This is a minor release which fixes two import issues. See [#9372](#) and [#9377](#). If you are not using BigTIFF or PerkinElmer .flex files, then you do not need to upgrade.

4.4.0 (July 2012)

This is a major release, which focuses on providing new functionality for controlling access to data, as well as significant improvements in our client applications.

The major theme of 4.4.0 is what we refer to as “Permissions”, the system by which users control access to their data. It is now possible to move data between groups, and much, much more.

We also added a few more things for users in 4.4.0, like:

- OMERO.insight webstart
- Importing from OMERO.insight is now complete
- Better integration of OMERO.insight with ImageJ
- A bottom-to-top reworking of the OMERO.web design

For developers and sysadmins, there are a few things as well:

- Support for Ice 3.4
- Removed support for PostgreSQL 8.3

Beta 4.3.4 (January 2012)

This is a point release is a security update to address an LDAP vulnerability.

Beta 4.3.3 (October 2011)

This point release is a short follow on to 4.3.2 to handle various issues found by users.

Beta 4.3.2 (September 2011)

This is a point release, focusing on fixes for OMERO.web, export, and documentation. A couple of LDAP fixes were also added, following requests from the community. We also included something many of you have asked for some time, OMERO on virtual machines.

Beta 4.3.1 (July 2011)

This point release focuses on fixes for Big Images, OMERO.web and others.

Beta 4.3.0 (June 2011)

This is a major release, focusing on new functionality for large, tiled images, and significant improvements in our client applications.

The major theme of 4.3.0 is what we refer to as “Big Images”, namely images with X,Y images larger than 4k x 4k. With this release, OMERO’s server and Java and web clients support tiling and image pyramids. This means we have the functionality you have probably seen in online map tools, ready for use in any image file format supported by OMERO (and obviously Bio-Formats). This is especially important for digital pathology, and other uses of stitched imaging.

While the major focus of 4.3.0 was Big Images, there are a number of other new updates. For users, we have worked hard to synchronise functionality and appearance across the OMERO clients. This includes viewing of ROIs in OMERO.web. We are not done, but we have made a lot of progress. Moreover, data import is now MUCH faster and available from within OMERO.insight.

Beta 4.2.2 (December 2010)

Fixes blocker reported using 4.2.1. Starting with this milestone, all tickets for the insight client are managed on Trac.

Beta 4.2.1 (November 2010)

This is a point release, focusing on fixes for delete functionality, and significant improvements in the way OMERO.web production server is deployed.

Beta 4.2.0 (July 2010)

This release is a major step for OMERO, enabling a number of critical features for a fully functional data management system:

- User and Group Permissions and data visibility between users
- updates to the OME SPW Model and improvements in HCS data visualisation
- SSL connection between OMERO clients and server;
- full scripting system, accessible from command line and within OMERO.insight, including Figure Export and FLIM Analysis
- ROIs generated in OMERO.insight stored on server
- extended use of OMERO.Tables for analysis results
- performance improvements for import and server-side import histories
- revamped, fully functional OMERO.web web browser interface
- upgrade of Backend libraries in OMERO.server

Beta 4.1.1 (December 2009)

This release fixes a series of small bugs in our previous Beta 4.1 release.

Beta 4.1 (October 2009)

Improved support for metadata, especially for confocal microscopy; OMERO supports all of the file formats enabled by Bio-Formats. Export to OME-TIFF and QuickTime/AVI/MPEG from OMERO. Various improvements to OMERO clients to improve workflow and use.

This release introduces OMERO.qa - a feedback mechanism, to allow us to communicate more effectively with our community. OMERO.qa supports uploading of problematic files, and tracking of responses to any user queries. Moreover, OMERO.qa includes a demo feature: in collaboration with Urban Liebel at Karlsruhe Institute of Technology, we are providing demo accounts for OMERO. Use the Demo link at qa to contact us if you are interested in this.

For users who have had problems with memory-based crashes in OMERO.insight, the new OpenGL-based ImageViewer may be of interest. Also, we are now taking advantage of our modeling of HCS data, and releasing our first clients that support Flex, MIAS, and InCell 1000 file formats. OMERO.dropbox has been substantially extended, and now supports all the file formats supported by OMERO.

Beta 4.0.1 (April 2009)

A quick patch release that fixes some bugs and adds some new functionality:

- Fixed Windows installation and updated docs.
- Bug fixes (scriptingEngine, importer).
- Fix .lif import, add Li-Cor 2D (OMERO does gels!).
- API .dv and OME .ome.tiff now supported by OMERO.fs.
- Support negative pixel values in Rendering Engine.
- Archived images are now fully supported in OMERO.
- OMERO.web merged with OmeroPy in distribution.

Beta 4.0 (March 2009)

This release consists of a major change in the remoting infrastructure, complete migration of existing OMERO clients to the ICE framework, two new OMERO clients, and integration of OMERO.editor into OMERO.insight.

OMERO.server updates:

- remove JBOSS, and switch all remoting to ICE
- improve session management, supporting creation of many thousands of session
- addition of an import service for server-side importing
- DB upgrades to support the metadata completion facilities
- substantial improvement to the interaction between the indexing engine and the rest of server.

OMERO.importer updates:

- migration to Blitz interface, giving much faster performance
- more efficient importing, complete metadata support for Zeiss LSM510, Leica LIF, Zeiss ZVI, Applied Precision DV, and MetaMorph STK

- addition of command line importer for batch import

OMERO.insight updates:

- migration to Blitz interface, giving much faster performance
- updates to metadata display, include complete support for OME Data Model
- much expanded integration of protocol management via OMEREditor, within OMERInsight
- support for image delete
- refinement of Projection Interface

OMERO.web: all new browser-based client for OMER. Enables sharing of images with colleagues with an account on server.

OMEREditor: a management tool for experimental protocols, now fully integrated with OMERInsight, so that protocols and experimental descriptions can be saved along with images and datasets. Includes a new parameters function, so that protocols in traditional documents can be easily imported into OMER. Supports, tables and .xls files. Also runs as a standalone application.

OMER.fs: a new OMER client, that monitors a specific directory and enables automatic imports. In its first incarnation, has quite limited functionality, supporting automatic import of LSM510 files only.

Beta 3.2 (November 2008)

The final update in the Beta3.x series. A number of fixes:

- faster thumbnailing and better support for large numbers of thumbnails
- improved handling of Leica .lei and Zeiss .zvi files
- extended support for reading OMEREditor files in OMERInsight
- measurement tool fixes in OMERInsight
- fixed memory problem in OMERInsight on Windows
- fixed thumbnailing and session bugs on OMER.server
- fixed DB upgrades for older PostgreSQL versions

Beta 3.0 (June 2008)

This release of OMER is a major update of functionality. In OMER.server, we have added support for StructuredAnnotations a flexible data management facility that allows essentially any kind of accessory data to be linked to images and experiments stored in OMER. Alongside this, we provide an indexing engine, that provides a flexible searching facility for essentially any text stored in an installation of OMER.server. Finally, we are releasing our first examples of clients that use the OMER.blitz server, a flexible, distributed interface that supports a range of client environments. One very exciting addition is OMER matlab, a gateway that can be used to access OMER from MATLAB®.

OMER Beta3.0 includes a substantial reworking of our clients as well. OMERInsight has been substantially updated, with an updated interface to provide a more natural workflow and support for many different types of annotations, through the StructuredAnnotations facility. The new search facilities are supported with smart user interfaces, with auto-complete, etc. New file formats have been added to OMER.importer, including support for OME-XML, and an improved import history facility is now available. Finally, Beta3.0 includes the first release of our experimental electronic notebook tool, OMEREditor. This represents our recent efforts to capture as much metadata around an experiment as possible.

Beta 2.3.3 insight (April 2008)

A new Beta 2.3.3 OMERO.insight has been released, this adds rotation to ellipse figure, and new format for saving intensity values.

Note: this version saves the ROIs in a format which is incompatible with previous saved ROIs.

Beta 2.3.1 importer (February 2008)

A new Beta 2.3.1 OMERO.importer has been released which includes a number of new formats: Zeiss AxioVision ZVI (Zeiss Vision Image), Nikon NIS-Elements .ND2 , Olympus FluoView FV1000, ICS (Image Cytometry Standard), PerkinElmer UltraView, and Jpeg2000.

The OMERO downloads for Beta 2.3 include a number of new options: a new import history feature, a Windows server installation, and a new tagging feature for OMERO.insight.

Note: milestone:3.0-Beta2.3 and prior Mac OS X installers for OMERO.server do not work on Mac OS X Leopard (10.5). Please follow the UNIX-based platform manual install instructions. Mac OS X installers for OMERO.insight and OMERO.importer work just fine under Leopard and can be used.

Beta 2.3 (December 2007)

This is a patch release for OMERO.server to fix a memory problem. In OMERO.insight, updating of the tagging facility, viewing of others' rendering settings and support for server-side compression of images before transport to client.

Beta 2.2 (November 2007)

In this release we have updated OMERO.server to run a newer version of JBOSS and provided support for copying display settings across a range of images. More new file formats. OMERO.insight has been updated to support copying display settings across many images. Image Viewer has been substantially updated.

Beta 2.1 (August 2007)

This is a client-only release. OMERO.insight now supports basic ROI measurements and a series of new file formats have been added. The OMERO downloads for Beta 2.0 have been simplified. OMERO.insight and OMERO.importer have been combined into a single download file called 'OMERO.clients' and the user documentation is now included inside of the server and client downloads.

Beta 2.0 (June 2007)

Note: this version will still work with the Beta 1 server release.

This major update provided our first support for multiple platforms via OMERO.Blitz. OMERO.insight now supports viewing work of multiple users. Beta 2 is our first release of the Web2.0-like 'tag' system developed in collaboration with Usable Image from Dundee University Computing department. This version addresses issues with using our tools under Java 1.6

Beta 1.1 (March 2007)

Patch release to fix time-out issues.

Beta 1 (January 2007)

The first public OMERO release, providing simple data management. Limited file format support (DV, STK, TIFF). Simple data visualization and management.

Milestone M3 (November 2006)

Rendering and compression API and client-side import. Access control and permissions system. Importer based on Bio-Formats.

Milestone M2 (July 2006)

The stateful rendering service is functional and all rendering code moved from Shoola Java client to the server. Also, the stateless services (IQuery,IUpdate,IPojos) are frozen and testing and documentation is checked and solidified.

Milestone M1 (April 2006)

Contains minimal functionality needed to run Shoola Java client without Perl server to demonstrate acceleration of metadata access. Application deployed on JBoss (<https://www.jboss.org>). No ACLs or permissions.

SYSTEM ADMINISTRATOR DOCUMENTATION

This documentation begins with information aimed at OS-level administrators and moves on to day-to-day management of OMERO for facility managers (who may find it useful to read the [Facility Managers walkthrough](#) for an overview first).

2.1 Getting started

The OMERO server system provides storage and processing of image data which conforms to the [OME Specification](#). It can be run on commodity hardware to provide your own storage needs, or run site-wide to provide a large-scale collaborative environment.

Although getting started with the server is relatively straightforward, it does require installing several software systems, and more advanced usage including backups and integrated logins, needs a knowledgeable system administrator.

2.1.1 Usage

You may find the [OMERO clients overview](#) user guide useful before working through the installation and maintenance guides provided in this section of the documentation.

2.1.2 Components

The server system is composed of several components, each of which runs in a separate process but is co-ordinated centrally.

- [OMERO.blitz](#) - the data server provides access to metadata stored in a relational database as well as the binary image data on disk.
- [OMERO.dropbox](#) - a filesystem watcher which notifies the server of newly uploaded or modified files and runs a fully automatic import (designed as the first implementation of [OMERO.fs](#) referred to in the architecture diagram).
- [OMERO.processor](#) - a process-launcher for running user-defined scripts.
- [OMERO.tables](#) - provide a way to efficiently store large, tabular results.
- [OMERO.indexer](#) - keeps a full-text search index up-to-date for searching.

If you are interested in building components for the server, modifying an existing component, or just looking for more background information, there is a section about the server within the [Developer Documentation](#); the best starting point is the [OMERO.server overview](#) for developers.

2.1.3 Background reading

What's new for OMERO 5.6 for sysadmins

- Server installation instructions for RHEL 9/RockyLinux 9 and Ubuntu 22.04. See [OMERO.server installation](#).
- Server configuration using self-signed certificates. We recommend to read [OMERO.server upgrade](#).
- [Version requirements](#) has been updated to reflect changes in version support for 5.6.0 and tentative plans for 6.0.0. The biggest change is of course the removal of support for Python 2.
- Installation walkthroughs now all suggest use of a virtualenv for installing Python packages. The upgrade guides will help with the migration. See [Migration to Python 3](#) for more information.
- See [the previous What's new page](#) for more details.

For a full list of bug fixes and other improvements, see the [CHANGELOGS](#).

Version requirements

Summary of changes for OMERO 5.6

We aim to support OMERO on the environments specified below, based on the availability of support by upstream developers and operating system distributions. This applies over the lifetime of the 5.6 release and includes security support. Support is limited to those environments on which OMERO is routinely tested.

This page details the minimum version requirements for the current (5.6) release.

It is intended to provide a roadmap in order that sysadmins may plan ahead and ensure that prerequisites are in place for future upgrades.

Level	Meaning
	unsupported/new
	supported/suboptimal
	supported/optimal
	supported/deprecated
	unsupported/old
	unsupported/broken
	unsupported/misc

Please check the full [support levels table](#) for more info on each support level.

Bitness

Rationale: OMERO is tested on 64-bit systems only.

Bitness	OMERO 5.6
32-bit	
64-bit	

NGINX

nginx	OMERO 5.6
1.23	
1.24	
1.25	

Operating system support

The following subsections detail the versions of each operating system which are supported by both its upstream developers (for security and general updates) and by OME for OMERO building and server deployment.

Linux (CentOS and RHEL)

General overview for [RHEL](#) and [CentOS](#).

Version	Release date	Supported until	OMERO 5.6
RHEL 7	June 2014	June 2024	
CentOS 7	June 2014	June 2024	
RHEL 8	May 2019	Dec 2029	
CentOS 8	May 2019	Dec 2021	
RHEL 9	May 2022	May 2032	
Rocky 9	Jul 2022	May 2032	

RHEL 9/Rocky Linux 9 are supported at present. Given the long life of enterprise releases, we intend to support only the latest release at any given time or else it ties us into very old dependencies.

Linux (Ubuntu)

General overview

Version	Release date	Supported until	OMERO 5.6
18.04 LTS	Apr 2018	Apr 2028	
20.04 LTS	Apr 2020	Apr 2030	
22.04 LTS	Apr 2022	Apr 2032	

Only the LTS releases are supported due to resource limitations upon CI and testing. Only the last LTS release is supported. There is currently no CI testing for any version.

Microsoft Windows

Client support only. See [blog post explanation](#)

MacOS X

MacOS X is typically suited only to client use, not serious server deployment, although the server can be expected to run on versions with current security support for testing purposes.

Dependencies

The following subsections detail the versions of each dependency needed by OMERO which are supported by both its upstream developers (for security and general updates) and by OME for OMERO building and server and client deployment.

Note: Versions in brackets are in development distributions and may change without notice.

Package lists

Operating system	Details
CentOS 7 / RHEL 7	Reference
Rocky 9	Reference
Ubuntu	Reference

PostgreSQL

General overview

OMERO support policies

Version	Release date	Supported until	OMERO 5.6
12	Sep 2019	Nov 2024	
13	Sep 2020	Nov 2025	
14	Sep 2021	Nov 2026	
15	Sep 2022	Nov 2027	
16	Sep 2023	Nov 2028	

Version provided by distribution

If no version is provided, a suitable repository is indicated.

Version	CentOS/RHEL	Ubuntu
12	9 (postgresql)	22.04 (postgresql)
13	9	22.04 (postgresql)
14	9 (postgresql)	22.04
15	9 (postgresql)	22.04 (postgresql)
16	9 (postgresql)	22.04 (postgresql)

The PostgreSQL project provides [packages](#) for supported platforms therefore distribution support is not necessary.

Python

OMERO support policies

Version	Release date	Supported until	OMERO 5.6	Details
3.6	Dec 2016	Dec 2021		PEP 494
3.7	Jun 2018	Jun 2023		PEP 537
3.8	Oct 2018	Oct 2024		PEP 569
3.9	Oct 2020	Oct 2025		PEP 596
3.10	Oct 2021	Oct 2026		PEP 619
3.11	Oct 2022	Oct 2027		PEP 664
3.12	Oct 2023	Oct 2028		PEP 693

Version provided by distribution

Version	CentOS/RHEL	Ubuntu
3.9	9	
3.10		22.04

The Django version used by OMERO.web (5.23.0) requires Python 3.8 or higher.

Ice

General overview

OMERO support policies

Version	Release date	Supported until	OMERO 5.6	Details
3.5	Mar 2013	Oct 2013		Release notes
3.6	June 2015	TBA		Release notes
3.7	July 2017	TBA		Release notes

Version provided by distribution

If no version is provided, a suitable repository is indicated.

Version	CentOS/RHEL	Ubuntu
3.6	9 (zeroc-ice-rhel9-x86_64)	22.04 (zeroc-ice-ubuntu2204-x86_64)

Java

[General overview](#)

OMERO support policies

Version	Release date	Supported until	OMERO 5.6	Details
7	Jul 2011	Apr 2015		Reference
8	Mar 2014	Nov 2026		Reference
11	Sep 2018	Oct 2024		Reference
17	Sep 2018	Oct 2027		Reference

Version provided by distribution

Version	CentOS/RHEL	Ubuntu
11	9	22.04
Details		Reference

Note that all distributions provide OpenJDK due to distribution restrictions by Oracle. [Oracle Java](#) may be used if downloaded separately.

NGINX

[General overview and roadmap](#)

OMERO support policies

Version	Release date	Supported until	OMERO 5.6
1.23	Jun 2022	May 2023	
1.24	Apr 2023	TBA	
1.25	May 2023	TBA	

Version provided by distribution

If no version is provided, a suitable repository is indicated.

Version	RHEL/Rocky Linux	Ubuntu
1.23	9 (repo)	N/A
1.24	9 (repo)	22.04

Support levels

The following table defines the symbols used throughout this page to describe the support status of a given component, as it progresses from being new and not supported, to supported and tested on a routine basis, and to finally being old and no longer supported nor tested.

Leve	Meaning	Description
	unsup-ported/new	New version not yet regularly tested and not officially supported; may or may not work (use at own risk)
	sup-ported/subop	Version which is tested, confirmed to work correctly, but may not offer optimal performance/experience
	sup-ported/optim	Version which is regularly tested, confirmed to work correctly, recommended for optimal performance/experience
	sup-ported/depre	Version which is less tested, expected to work correctly, but may not offer optimal performance/experience; official support may be dropped in the next major OMERO release
	unsup-ported/old	Old version no longer tested and no longer officially supported; may or may not work (use at own risk)
	unsup-ported/broke	Known to not work
	unsup-ported/misc	Not supported for some reason other than the above

System requirements

Hardware

OMERO.server

The system requirements for OMEROServer vary greatly depending on image size and number of users. The minimum requirements should be easily exceeded by any recently bought hardware.

An OMEROServer specification for between 25-50 users might be:

- Quad-core Intel or AMD CPU
- 8 GB RAM
- 256 GB solid-state drive space for OMEROServer distribution
- Hard drive space proportional to the image sizes expected (likely between 10 and 100 TB)

A specification for a server future-proofed for 3-4 years might be:

- Intel Xeon Gold 6426 Processor
- 8x 32 GB DDR5-4800
- 2 x 2 TB SSD RAID1 for OS, PostgreSQL DB, scratch, log files, etc.
- 10 GbE connectivity to a separate fileshare for the OMEROServer binary repository

Storage

Hard drive space should be proportional to the image sizes expected. The drive space should permit **proper locking**, which is often not the case with remotely mounted shares. See the [binary repository](#) section for more information.

RAM

RAM is not going to scale linearly, particularly with the way the JVM works. You are probably going to hit a hard ceiling between 4 and 6 GB for JVM size (there is really not much point in having it larger anyway). With a large database and aggressive PostgreSQL caching your RAM usage could be larger. Still, even for a large deployment, it is not cost effective to use more than a few GBs of RAM for this purpose. [Performance and monitoring](#) provides information about fine-tuning the server processes' memory usage. In summary, depending on hardware layout 16, 24 or 32 GB of RAM would be ideal for your OMEROServer. If you have a separate database server more than 16 GB of RAM may not be of much benefit to you at all.

CPU

CPU is not something that an OMEROSystem is usually ever limited by. However, when it is limited, it is almost always limited by [single thread performance](#) and not by the CPU core count. A 16 system core count should be more than enough. You are not going to get a huge OMEROServer performance increase by, for example, throwing 24 cores at the problem; a specification with a focus on higher single thread performance is going to give you better performance.

Further examples

Example production server set-ups provides details on some production set-ups in use by OMERO admins, along with how many users and the amount of data they support, which you may find helpful.

OMERO.insight and OMERO.importer

The recommended client specification is:

- Intel or AMD CPU
- 8 GB RAM
- 120 GB solid-state drive (SSD)

Client configuration

When performing some operations the clients make use of temporary file storage and log directories. The table below indicates the default values for each directory and the environment variables for overriding their locations:

Client directory	Environment variable	Default location (UNIX)	Default location (Windows)
OMERO user directory	OMERO_USERDIR	\$HOME/omero	%HOMEPATH%\omero
Temporary files	OMERO_TMPDIR	\$HOME/omero/tmp	%HOMEPATH%\omero\tmp
Local sessions	OMERO_SESSIONDIR	\$HOME/omero/ sessions	%HOMEPATH%\omero\ sessions
Log files		\$HOME/omero/log	%HOMEPATH%\omero\log

Note that setting OMERO_USERDIR will also change the default location for the temporary files and the local sessions.

If your home directory is stored on a network, possibly NFS mounted (or similar), then these temporary files are being written and read over the network. This can slow access down.

See also:

Troubleshooting performance issues with the clients

Troubleshooting section about client performance issues on NFS

Software

Each component of the OMERO platform has a separate set of prerequisites. Where possible, we provide tips on getting started with each of these technologies, but we can only provide free support within limits.

Package	OMERO.server	Java	Python	Ice	PostgreSQL
OMERO.importer	Required	Required			
OMERO.insight	Required	Required			
OMERO.server		Required	Required	Required	Required
OMERO.web	Required		Required	Required	
OMERO.py	Required for some functionality		Required	Required	
OMERO.cpp	Required for some functionality			Required	

For full details on which versions of these are supported for OMERO 5.6 and how we intend to update these going forward, see the [Version requirements](#) section.

Example production server set-ups

CellNanOs (Center of Cellular Nanoanalytics), University of Osnabrück

The OMERO server at [CellNanOs](#) serves a community of 75-100 users and 17 microscope stations (13 different systems), producing 180-1360 GB of data per day. It is hosted on RedHat 7.3 with data stored on an IBM GPFS file system.

Hardware

- Dell R630 running RedHat 7.3, 32 cores, 128 GB RAM
- IBM GPFS storage, 6 TB SSDs, 178 TB SATA

Network infrastructure

1-10 GBit connection between microscope workstations and OMERO

Backup/archive

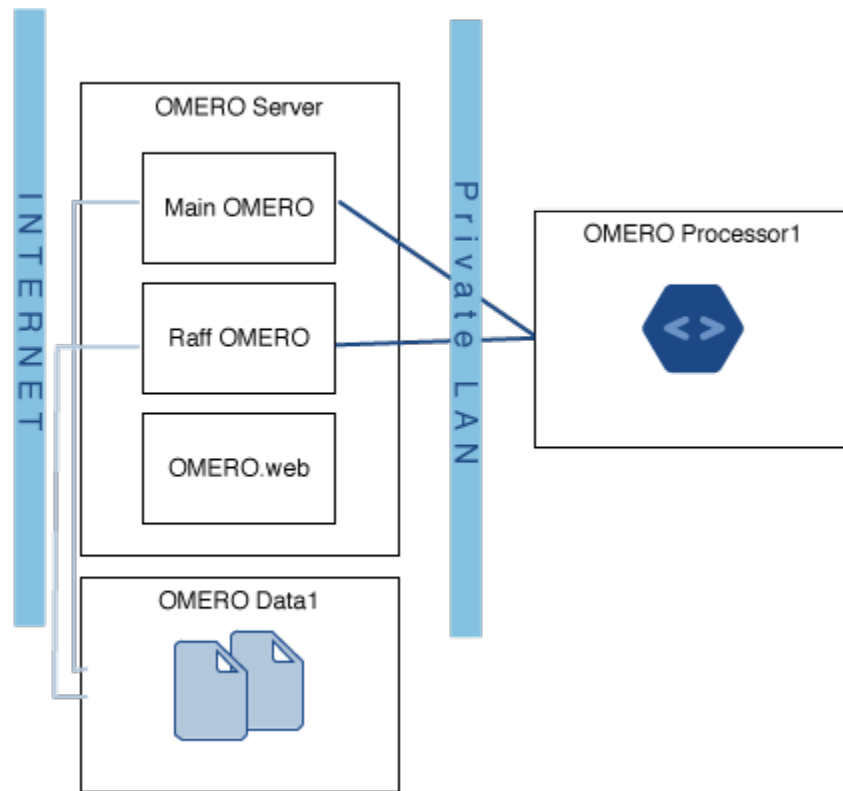
- IBM TSM 1.4 PB
- daily migration of new data to tape, archive on tape

Micron, Oxford

The OMERO server at [Micron, Oxford](#) houses two OMERO instances, the databases for both these instances, and a single OMERO.web instance which serves them both. The second OMERO instance (Raff OMERO) originated from another group's private OMERO server, which is now managed by Micron, but there was no way to merge this data into the main server. The main OMERO instance is configured to interface to a departmental LDAP server to authenticate users and get initial authorization details.

OMERO Data1 in the diagram is a large filestore server which hosts all the image data. This is made available to the OMERO server itself via a Samba mount. This server has 36 TiB of space of which OMERO is using 16 TiB and Raff OMERO is using 600 GiB. This is backed up to a tape robot.

OMERO Processor1 consists of a 32 core, 128GiB RAM processing machine for doing image analysis. This is connected on a completely private network to the OMERO server (to avoid issues with configuring OMERO.grid to be secure) and runs scripts using OMERO.grid.



Stats

- 90 users
- 40 groups
- 36 TiB of data storage space, of which 16.6 TiB is currently in use
- Performance statistics to come

IMCF, Biozentrum, University of Basel

The OMERO server at the [IMCF / Biozentrum](#) has around 650 users and uses more than 200 TB of data storage space, with an average monthly increase of 10 TB (as of mid-2021). It is run on CentOS 7 with data hosted on a native-mounted GPFS file system.

Hardware

Remote storage consists of:

- native-mounted GPFS volume

Local storage consists of:

- 2 x 240 GB SATA SSD, RAID 1, OS and OMERO software
- 2 x 400 GB SATA SSD, RAID 1, Postgres DB

Computational resources:

- Lenovo System x3650 M5
- 12 Cores (2 x Intel Xeon E5-2643v3 3.4GHz)
- 256 GB RAM

Network infrastructure

- 40 Gbit/s Infiniband connection to GPFS storage
- 10 Gbit/s Ethernet connection to the client network

GReD Research Center, Clermont-Ferrand, France

The [Genetics, Reproduction and Development Research Center](#) has 65 users and currently uses 3 TB of storage, with an average monthly increase of 90 GB. It is run on Debian Squeeze.

Hardware

- 11 TB of storage spread over 8 local hard drives (2 TB), RAID 5

Computational resources:

- 1 Intel Xeon E5506 (4 physical cores)
- 8 GB of memory

Network infrastructure

The server is hosted inside the faculty of medicine where the network works at 100 Mbit/s. There are 4 Gbit/s ports on the server but only one is currently in use.

Image Data Resource

The [Image Data Resource](#) is an OMERO repository maintained by OME and deployed on the EMBL-EBI Embassy Cloud which publishes reference imaging datasets. See the [IDR deployment page](#) for more information about the architecture and the [IDR studies page](#) for the most up-to-date metrics.

Known limitations

Time zone

We do not recommend changing the time zone on your server. The server is currently set to use local time and changing time zones will result in a mismatch between the original data import times stored in the server and the way the clients report them.

Too many open file descriptors

Starting with OMERO 5, the server works directly from original files. At times, this requires a significant number of open file handles. If you are having problems with large or frequent imports, are seeing “Too many open file descriptors” or similar, you may need to increase the maximum number of open files per process. On Linux, this may be done by setting the *nofile* limit in `/etc/security/limits.conf`, for example:

```
omero soft nofile 10000
omero hard nofile 12000
```

This permits the *omero* user to have 10000 open files per process, which may be increased up to a maximum of 12000 by the user. The username and limits will need adjusting for the specifics of your installation and usage requirements. Note that these settings take effect only for new logins, so the server and the shell or environment the server is started from will require restarting. Run `ulimit -a` as the user running OMERO to verify that the changes have taken effect.

Changing group permissions

If a group contains a projection made by one member from data owned by another user, you cannot make the group into a private group.

File format support

Large images

When you import an image over a certain size, OMERO will generate a pyramid of lower resolution images if it doesn't already exist in the file. The threshold size is configurable using `omero.pixeldata.max_plane_height` and `omero.pixeldata.max_plane_width` but set to 3192x3192 pixels by default. However, this process can be very resource-intensive, depending on the size of the image as well as the image format and any data compression used, for example see [PixelData threads and pyramid generation issues](#).

Further, OMERO never generates pyramids for large floating-point pixel type images.

For large floating-point images, follow the recommended workflow to have the best experience (see below).

The OMERO pyramid generation process should be considered as deprecated and instead it is recommended that users avoid these issues by converting their data to [pyramidal OME-TIFF](#) files before importing into OMERO. A number of suitable tools are available such as [bioformats2raw](#) & [raw2ometiff](#), [bfconvert](#), [Kheops](#), [tiff file](#), [aicsimageio](#), [libvips](#) and [QuPath](#).

Large floating-point images

[Pyramids](#) of image tiles are currently not generated for large floating-point pixel type images.

This primarily affects the following file formats:

- [Gatan DM3](#)
- [MRC](#)
- [TIFF](#)

However, in some cases, the floating-point images without [pyramids](#) can be viewed in OMERO clients at full resolution (if the images are not too large).

This behaviour is configured by `omero.pixeldata.max_plane_float_override`. By default (True), OMERO overrides the requirement for floating-point images above the `omero.pixeldata.max_plane_height` and `omero.pixeldata.max_plane_width` to have pyramids, which allows them to be treated as regular images and possibly viewed in the clients.

However, this also allows OMERO to attempt the calculation of minimal and maximal pixel intensity for these images (normally disabled for large images because it is resource intensive to read every pixel value).

When the `omero.pixeldata.max_plane_float_override` is set to False on your server, OMERO will not attempt to treat large floating-point images as if they are smaller images, so any large images without pre-generated pyramids will not be viewable. However, this will protect the server from expensive attempts to calculate min/max pixel values.

It is recommended to pre-generate pyramidal OME-TIFF images as described above and to set `omero.pixeldata.max_plane_float_override` to False on your server.

Import of OME-NGFF

The import of OME-NGFF is currently limited to the *command-line (CLI) importer* only.

Naming of OME-NGFF images in OMERO

The default naming of the OME-NGFF Images imported into OMERO is not intuitive at the moment. Use the `omero import -n` option to achieve explicit naming.

Depth of scanning prior to import

The import might fail if the hierarchy of folders is exceeding the depth of scanning (default: 4). For formats using deeper hierarchy of folders such as OME-NGFF use `omero import --depth` option to set the depth of scanning of 10 (or more if necessary).

Calculation of minima and maxima pixel values

If images are imported with one of the `omero import --skip` options skipping calculation of the global minima and maxima pixel values, OMERO clients will use the extrema of the pixel type range by default. Users can adjust the minima/maxima via the rendering settings. Recalculating minima and maxima pixel values after import is currently not supported.

Flex data in OMERO.tables

If you are using the advanced configuration setting `FlexReaderServerMaps` for importing Flex data split between multiple directories for use with *OMERO.tables*, you should not upgrade beyond 5.0.x. Neither the 5.1 line nor OMERO 5.2 support this functionality.

LDAP

Enabling synchronization of LDAP on user login may override admin actions carried out in the clients, see *Synchronizing LDAP on user login* for details.

2.2 Installation

This section provides guidance on how to install and set up OMERO.server and OMERO.web on any of the supported UNIX and UNIX-like platforms. Following the installation links below you will find specific walkthroughs provided for several systems, with detailed step-by-step instructions. Reading through the *OMERO.server installation* and *OMERO.web installation and maintenance* pages first is recommended as this explains the entire process rather than just being a series of commands.

2.2.1 OMERO.server installation

This section covers the installation of OMERO.server on UNIX and UNIX-like platforms. This includes all BSD, Linux and Mac OS X systems. Depending upon which platform you are using, you may find a more specific walk-through listed below but we recommend you read through this page first as it explains the entire process rather than just being a series of commands. The walk-throughs describes how to install the **recommended** versions, not all the supported versions. This should be read in conjunction with *Version requirements*.

Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process.

Recommended:

OMERO.server installation on RHEL 9/Rocky Linux 9

Instructions for installing OMERO.server from scratch on RHEL 9/Rocky Linux 9 with Ice 3.6 and Python 3.9.

OMERO.server installation on Ubuntu 22.04

Instructions for installing OMERO.server from scratch on Ubuntu 22.04 with Ice 3.6 and Python 3.10.

Supported:

OMERO.server installation on CentOS 7

Instructions for installing OMERO.server from scratch on CentOS 7 with Ice 3.6 and Python 3.6.

OMERO.server installation on Ubuntu 20.04

Instructions for installing OMERO.server from scratch on Ubuntu 20.04 with Ice 3.6 and Python 3.8.

Development:

OMERO.server installation on OS X with Homebrew

Instructions for installing and building OMERO.server on Mac OS X with dependencies installed using Homebrew. It is aimed at **developers** since typically MacOS X is not suited for serious server deployment.

OMERO.server installation on CentOS 7

Warning: CentOS 7 will reach End Of Life on June 30, 2024. We recommend to use RHEL 9/Rocky 9 instead. Please read [OMERO.server installation on RHEL 9/Rocky Linux 9](#).

This is an example walkthrough for installing OMERO on CentOS 7, using a dedicated local system user. You can use this as a guide for setting up your own test server. For production use you should also read the pages listed under [Optimizing Server Configuration](#). This guide will install Python 3.6. Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process.

This guide describes how to install using the **recommended** versions for Java, Ice, PostgreSQL. This should be read in conjunction with [Version requirements](#).

This guide **does not** describe how to install OMERO.web. To deploy OMERO.web, please read [OMERO.web installation on CentOS 7 and IcePy 3.6](#).

These instructions assume your Linux distribution is configured with a UTF-8 locale (this is normally the default).

For convenience in this walkthrough, we will use the **omero-server system user** and the main OMERO configuration options have been defined as environment variables. When following this walkthrough you can either use your own values, or alternatively create `settings.env` for example under `/tmp` e.g. `/tmp/settings.env` containing the variables below and source it when required:

```
OMERO_DB_USER=db_user
OMERO_DB_PASS=db_password
OMERO_DB_NAME=omero_database
OMERO_ROOT_PASS=omero_root_password
OMERO_DATA_DIR=/OMERO

export OMERO_DB_USER OMERO_DB_PASS OMERO_DB_NAME OMERO_ROOT_PASS OMERO_DATA_DIR

export PGPASSWORD="$OMERO_DB_PASS"

# Location of the OMERO.server
export OMERODIR=/opt/omero/server/OMERO.server

# Location of the virtual environment for omero-py
VENV_SERVER=/opt/omero/server/venv3

export ICE_HOME=/opt/ice-3.6.5

export PATH=$ICE_HOME/bin:$VENV_SERVER/bin:$PATH
```

Installing prerequisites

The following steps are run as root.

Install Java 11, Ice 3.6.5 and PostgreSQL 14:

To install Java 11 and other dependencies:

```
yum -y install epel-release

yum -y install unzip wget bc

# install Java
yum -y install java-11-openjdk

# install dependencies

yum -y install python3
yum -y install openssl
```

To install Ice 3.6.5:

```
curl -sL https://zeroc.com/download/Ice/3.6/el7/zeroc-ice3.6.repo > \
/etc/yum.repos.d/zeroc-ice3.6.repo

yum -y install glacier2 \
icebox \
icegrid \
icepatch2 \
libfreeze3.6-c++ \
libice3.6-c++ \
libicestorm3.6
```

To install PostgreSQL 14:

```
yum -y install https://download.postgresql.org/pub/repos/yum/reporepms/EL-7-x86_64/pgdg-
redhat-repo-latest.noarch.rpm
yum -y install postgresql11-server postgresql11

PGSETUP_INITDB_OPTIONS=--encoding=UTF8 /usr/pgsql-11/bin/postgresql-11-setup initdb

sed -i.bak -re 's/^(host.*)ident/\1md5/' /var/lib/pgsql/11/data/pg_hba.conf
systemctl start postgresql-11.service

systemctl enable postgresql-11.service
```

Note: if you are installing PostgreSQL in a Docker container, some of the commands above will not work. For more details check [step01_centos7_pg_deps.sh](#)

Create a local omero-server system user, and a directory for the OMERO repository:

```
useradd -mr omero-server
# Give a password to the omero user
```

(continues on next page)

(continued from previous page)

```
# e.g. passwd omero-server
chmod a+X ~omero-server

mkdir -p "$OMERO_DATA_DIR"
chown omero-server "$OMERO_DATA_DIR"
```

Make the `settings.env` available to the `omero-server` system user by copying in to the user home directory. The file will need to be sourced each time you switch user. You could add `. ~/settings.env` to the `omero-server` system user bash profile.

Create a database user and initialize a new database for OMERO:

```
echo "CREATE USER $OMERO_DB_USER PASSWORD '$OMERO_DB_PASS'" | su - postgres -c psql
su - postgres -c "createdb -E UTF8 -O '$OMERO_DB_USER' '$OMERO_DB_NAME'"

psql -P pager=off -h localhost -U "$OMERO_DB_USER" -l
```

Installing OMERO.server

The following step is run as root.

We recommend to create a virtual environment and install the Ice Python binding and the dependencies required by the server using pip:

```
# Create a virtual env
python3 -mvenv $VENV_SERVER

# Upgrade pip
$VENV_SERVER/bin/pip install --upgrade pip

# Install the Ice Python binding
$VENV_SERVER/bin/pip install https://github.com/ome/zeroc-ice-py-centos7/releases/
↳download/0.2.1/zeroc_ice-3.6.5-cp36-cp36m-linux_x86_64.whl

# Install server dependencies
$VENV_SERVER/bin/pip install omero-server
```

Download and unzip OMERO.server:

```
cd /opt/omero/server
SERVER=https://downloads.openmicroscopy.org/omero/5.6/server-ice36.zip
wget -q $SERVER -O OMERO.server-ice36.zip
unzip -q OMERO.server*
```

Change the ownership of the OMERO.server directory and create a symlink:

```
# change ownership of the folder
chown -R omero-server OMERO.server-*
ln -s OMERO.server-* / OMERO.server
```

Configuring OMERO.server

The following steps are run as the `omero-server` system user. (`su - omero-server`)

The variable `OMERODIR` set in `settings.env` above **must** point to the location where OMERO.server is installed. e.g. `OMERODIR=/path_to_omero_server/OMERO.server`.

Note that this script requires the same environment variables that were set earlier in `settings.env`, so you may need to copy and/or source this file as the `omero` user.

Configure the database and the location of the data directory:

```
omero config set omero.data.dir "$OMERO_DATA_DIR"
omero config set omero.db.name "$OMERO_DB_NAME"
omero config set omero.db.user "$OMERO_DB_USER"
omero config set omero.db.pass "$OMERO_DB_PASS"
omero db script -f $OMERODIR/db.sql --password "$OMERO_ROOT_PASS"
psql -h localhost -U "$OMERO_DB_USER" "$OMERO_DB_NAME" < $OMERODIR/db.sql
```

Weaker ciphers like ADH are disabled by default in new versions of OpenSSL and TLS versions 1.0 and 1.1 have been dropped from JDK packages. In order to connect to an OMERO.server using any OMERO clients e.g. the Java Desktop client, the OMERO.web client or the CLI and import data, you need to generate self-signed certificates after installing the [omero-certificates](#) package.

```
omero certificates
```

Additionally on CentOS7, it is necessary to prevent the OMERO.server from advertising Diffie-Hellmann key exchange to the clients. While this key-exchange algorithm is disabled in the OMERO side, if it is advertised it can lead to client errors (e.g. `omero-py` >= 5.13.0), which will fail to connect to OMERO because of a `dh-key` error. To prevent this, it might be necessary to remove Diffie-Hellmann key exchange from the IceSSL configuration. On CentOS7, this can be done by running the following code:

```
omero config set omero.glacier2.IceSSL.Ciphers=HIGH:!DH
```

See also [Client Server SSL verification](#).

Running OMERO.server

The following steps are run as the `omero-server` system user. (`su - omero-server`)

OMERO should now be set up. To start the server run:

```
omero admin start
```

Should you wish to start OMERO automatically, a *systemd service file* could be created. An example `omero-server-systemd.service` is available.

Copy the `systemd.service` file and configure the service:

```
cp omero-server-systemd.service /etc/systemd/system/omero-server.service

systemctl daemon-reload

systemctl enable omero-server.service
```

You can then start up the service.

Securing OMERO

The following steps are run as root.

If multiple users have access to the machine running OMERO you should restrict access to OMERO.server's configuration and runtime directories, and optionally the OMERO data directory:

```
chmod go-rwx $OMERODIR/etc $OMERODIR/var

# Optionally restrict access to the OMERO data directory
# chmod go-rwx "$OMERO_DATA_DIR"
```

OMERO.server installation on RHEL 9/Rocky Linux 9

This is an example walkthrough for installing OMERO on RHEL 9/Rocky Linux 9, using a dedicated local system user. You can use this as a guide for setting up your own test server. For production use you should also read the pages listed under *Optimizing Server Configuration*. This guide will install Python 3.9. Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process.

This guide describes how to install using the **recommended** versions for Java, Ice, PostgreSQL. This should be read in conjunction with *Version requirements*.

This guide **does not** describe how to install OMERO.web. To deploy OMERO.web, please read *OMERO.web installation on Rocky Linux 9 and IcePy 3.6*.

These instructions assume your Linux distribution is configured with a UTF-8 locale (this is normally the default).

For convenience in this walkthrough, we will use the **omero-server system user** and the main OMERO configuration options have been defined as environment variables. When following this walkthrough you can either use your own values, or alternatively create `settings.env` for example under `/tmp` e.g. `/tmp/settings.env` containing the variables below and source it when required:

```
OMERO_DB_USER=db_user
OMERO_DB_PASS=db_password
OMERO_DB_NAME=omero_database
OMERO_ROOT_PASS=omero_root_password
OMERO_DATA_DIR=/OMERO

export OMERO_DB_USER OMERO_DB_PASS OMERO_DB_NAME OMERO_ROOT_PASS OMERO_DATA_DIR

export PGPASSWORD="$OMERO_DB_PASS"

# Location of the OMERO.server
export OMERODIR=/opt/omero/server/OMERO.server

# Location of the virtual environment for omero-py
VENV_SERVER=/opt/omero/server/venv3

export ICE_HOME=/opt/ice-3.6.5

export PATH=$ICE_HOME/bin:$VENV_SERVER/bin:$PATH
```

Installing prerequisites

The following steps are run as root.

Install Java 11, Ice 3.6.5 and PostgreSQL 15:

To install Java 11 and other dependencies:

```
dnf -y install java-11-openjdk

# install dependencies
dnf -y install python unzip bzip2 wget bc openssl
```

To install Ice 3.6.5:

```
if grep -q "Rocky" /etc/redhat-release; then
    dnf -y install 'dnf-command(config-manager)'
    dnf config-manager --set-enabled crb
fi
if grep -q "Red Hat" /etc/redhat-release; then
    subscription-manager repos --enable codeready-builder-for-rhel-9-$(arch)-rpms
fi
dnf -y install expat libdb-cxx

cd /tmp
wget https://github.com/glencoesoftware/zeroc-ice-rhel9-x86_64/releases/download/
↪20231130/Ice-3.6.5-rhel9-x86_64.tar.gz
tar xf Ice-3.6.5-rhel9-x86_64.tar.gz
mv Ice-3.6.5 /opt/ice-3.6.5
echo /opt/ice-3.6.5/lib64 > /etc/ld.so.conf.d/ice-x86_64.conf
ldconfig
```

To install PostgreSQL 15:

```
dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-9-x86_64/pgdg-
↪redhat-repo-latest.noarch.rpm
dnf -qy module disable postgresql
dnf -y install postgresql15-server postgresql15
PGSETUP_INITDB_OPTIONS=--encoding=UTF8 /usr/pgsql-15/bin/postgresql-15-setup initdb

sed -i.bak -re 's/^(host.*)ident/\1md5/' /var/lib/pgsql/15/data/pg_hba.conf
sed -i 's/ ident/ trust/g' /var/lib/pgsql/15/data/pg_hba.conf
systemctl start postgresql-15
systemctl enable postgresql-15
```

Note: if you are installing PostgreSQL in a Docker container, some of the commands above will not work. For more details check [step01_rocky9_deps.sh](#)

Create a local omero-server system user, and a directory for the OMERO repository:

```
useradd -mr omero-server
# Give a password to the omero user
# e.g. passwd omero-server
```

(continues on next page)

(continued from previous page)

```
chmod a+X ~omero-server
mkdir -p "$OMERO_DATA_DIR"
chown omero-server "$OMERO_DATA_DIR"
```

Make the `settings.env` available to the `omero-server` system user by copying in to the user home directory. The file will need to be sourced each time you switch user. You could add `. ~/settings.env` to the `omero-server` system user bash profile.

Create a database user and initialize a new database for OMERO:

```
echo "CREATE USER $OMERO_DB_USER PASSWORD '$OMERO_DB_PASS'" | su - postgres -c psql
su - postgres -c "createdb -E UTF8 -O '$OMERO_DB_USER' '$OMERO_DB_NAME'"

psql -P pager=off -h localhost -U "$OMERO_DB_USER" -l
```

Installing OMERO.server

The following step is run as root.

We recommend to create a virtual environment and install the Ice Python binding and the dependencies required by the server using `pip`:

```
# Create a virtual env
python3 -mvenv $ENV_SERVER

# Upgrade pip
$ENV_SERVER/bin/pip install --upgrade pip

# Install the Ice Python binding
$ENV_SERVER/bin/pip install https://github.com/glencoesoftware/zeroc-ice-py-rhel9-x86_
↳ 64/releases/download/20230830/zeroc_ice-3.6.5-cp39-cp39-linux_x86_64.whl
# Install server dependencies
$ENV_SERVER/bin/pip install omero-server
```

Download and unzip OMERO.server:

```
cd /opt/omero/server
SERVER=https://downloads.openmicroscopy.org/omero/5.6/server-ice36.zip
wget -q $SERVER -O OMERO.server-ice36.zip
unzip -q OMERO.server*
```

Change the ownership of the OMERO.server directory and create a symlink:

```
# change ownership of the folder
chown -R omero-server OMERO.server-*
ln -s OMERO.server-* / OMERO.server
```

Configuring the OMERO.server

The following steps are run as the `omero-server` system user. (`su - omero-server`)

The variable `OMERODIR` set in `settings.env` above **must** point to the location where OMERO.server is installed. e.g. `OMERODIR=/path_to_omero_server/OMERO.server`.

Note that this script requires the same environment variables that were set earlier in `settings.env`, so you may need to copy and/or source this file as the `omero` user.

Configure the database and the location of the data directory:

```
omero config set omero.data.dir "$OMERO_DATA_DIR"
omero config set omero.db.name "$OMERO_DB_NAME"
omero config set omero.db.user "$OMERO_DB_USER"
omero config set omero.db.pass "$OMERO_DB_PASS"
omero db script -f $OMERODIR/db.sql --password "$OMERO_ROOT_PASS"
psql -h localhost -U "$OMERO_DB_USER" "$OMERO_DB_NAME" < $OMERODIR/db.sql
```

Weaker ciphers like ADH are disabled by default in new versions of OpenSSL and TLS versions 1.0 and 1.1 have been dropped from JDK packages. In order to connect to an OMERO.server using any OMERO clients e.g. the Java Desktop client, the OMERO.web client or the CLI and import data, you need to generate self-signed certificates after installing the [omero-certificates](#) package.

```
omero certificates
```

Running the OMERO.server

The following steps are run as the `omero-server` system user. (`su - omero-server`)

OMERO should now be set up. To start the server run:

```
omero admin start
```

Should you wish to start OMERO automatically, a *systemd service file* could be created. An example `omero-server-systemd.service` is available.

Copy the `systemd.service` file and configure the service:

```
cp omero-server-init.d /etc/init.d/omero-server
chmod a+x /etc/init.d/omero-server

update-rc.d -f omero-server remove
update-rc.d -f omero-server defaults 98 02
```

You can then start up the service.

Securing OMERO

The following steps are run as root.

If multiple users have access to the machine running OMERO you should restrict access to OMERO.server's configuration and runtime directories, and optionally the OMERO data directory:

```
chmod go-rwx $OMERODIR/etc $OMERODIR/var

# Optionally restrict access to the OMERO data directory
# chmod go-rwx "$OMERO_DATA_DIR"
```

OMERO.server installation on Ubuntu 20.04

Warning: We recommend to use Ubuntu 22.04. Please read [OMERO.server installation on Ubuntu 22.04](#).

This is an example walkthrough for installing OMERO on Ubuntu 20.04, using a dedicated local system user. You can use this as a guide for setting up your own test server. For production use you should also read the pages listed under [Optimizing Server Configuration](#). This guide will install Python 3.8. Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process.

This guide describes how to install using the **recommended** versions for Java, Ice, PostgreSQL. This should be read in conjunction with [Version requirements](#).

This guide does not describe how to install OMERO.web. To deploy OMERO.web, please read [OMERO.web installation on Ubuntu 20.04 and IcePy 3.6](#).

These instructions assume your Linux distribution is configured with a UTF-8 locale (this is normally the default).

For convenience in this walkthrough we will use the **omero-server system user** and the main OMERO configuration options have been defined as environment variables. When following this walkthrough you can either use your own values, or alternatively create settings.env for example under /tmp e.g. /tmp/settings.env containing the variables below and source it when required:

```
OMERO_DB_USER=db_user
OMERO_DB_PASS=db_password
OMERO_DB_NAME=omero_database
OMERO_ROOT_PASS=omero_root_password
OMERO_DATA_DIR=/OMERO

export OMERO_DB_USER OMERO_DB_PASS OMERO_DB_NAME OMERO_ROOT_PASS OMERO_DATA_DIR

export PGPASSWORD="$OMERO_DB_PASS"

# Location of the OMERO.server
export OMERODIR=/opt/omero/server/OMERO.server

# Location of the virtual environment for omero-py
VENV_SERVER=/opt/omero/server/venv3

export ICE_HOME=/opt/ice-3.6.5
```

(continues on next page)

(continued from previous page)

```
export PATH=$ICE_HOME/bin:$ENVV_SERVER/bin:$PATH
```

Installing prerequisites

The following steps are run as root.

Install Java 11, Ice 3.6.5 and PostgreSQL 12:

To install Java 11 and other dependencies:

```
apt-get update

apt-get -y install unzip wget bc

# to be installed if daily cron tasks are configured
apt-get -y install cron

# install Java
apt-get update -q
apt-get install -y openjdk-11-jre

# install dependencies
```

To install Ice 3.6.5:

```
apt-get install -y -q \
build-essential \
db5.3-util \
libbz2-dev \
libdb++-dev \
libdb-dev \
libexpat-dev \
libmcpp-dev \
libssl-dev \
mcpp \
zlib1g-dev

cd /tmp
wget -q https://github.com/ome/zeroc-ice-ubuntu2004/releases/download/0.2.0/ice-3.6.5-0.2.0-ubuntu2004-amd64.tar.gz
tar xf ice-3.6.5-0.2.0-ubuntu2004-amd64.tar.gz
mv ice-3.6.5-0.2.0 ice-3.6.5
mv ice-3.6.5 /opt
echo /opt/ice-3.6.5/lib64 > /etc/ld.so.conf.d/ice-x86_64.conf
ldconfig
```

To make Ice available to all users and activate the virtual environment, set the following in `/etc/profile`:

```
# Environment file for OMERO
```

(continues on next page)

(continued from previous page)

```
export ICE_HOME=/opt/ice-3.6.5
export PATH="$ICE_HOME/bin:$PATH"
#Remove commented out export below if Ice is not set globally accessible
#export LD_LIBRARY_PATH="$ICE_HOME/lib64:$ICE_HOME/lib:$LD_LIBRARY_PATH"
export SLICEPATH="$ICE_HOME/slice"
```

and add the virtual environment to PATH:

```
VENV_SERVER=/opt/omero/server/venv3

export ICE_HOME=/opt/ice-3.6.5

export PATH=$ICE_HOME/bin:$VENV_SERVER/bin:$PATH
```

To install PostgreSQL 12:

```
apt-get -y install postgresql
service postgresql start
```

Create a local omero-server system user, and a directory for the OMERO repository:

```
useradd -mr omero-server
# Give a password to the omero user
# e.g. passwd omero-server
chmod a+X ~omero-server

mkdir -p "$OMERO_DATA_DIR"
chown omero-server "$OMERO_DATA_DIR"
```

Make the `settings.env` available to the omero-server system user by copying in to the user home directory. The file will need to be sourced each time you switch user. You could add `. ~/settings.env` to the omero-server system user bash profile.

Create a database user and initialize a new database for OMERO:

```
echo "CREATE USER $OMERO_DB_USER PASSWORD '$OMERO_DB_PASS'" | su - postgres -c psql
su - postgres -c "createdb -E UTF8 -O '$OMERO_DB_USER' '$OMERO_DB_NAME'"

psql -P pager=off -h localhost -U "$OMERO_DB_USER" -l
```

Installing OMERO.server

The following step is run as root.

We recommend to create a virtual environment and install the Ice Python binding and the dependencies required by the server using pip:

```
# Create a virtual env
python3 -mvenv $VENV_SERVER

# Upgrade pip
$VENV_SERVER/bin/pip install --upgrade pip
```

(continues on next page)

(continued from previous page)

```
# Install the Ice Python binding
$ENV_SERVER/bin/pip install https://github.com/ome/zeroc-ice-ubuntu2004/releases/
↳download/0.2.0/zeroc_ice-3.6.5-cp38-cp38-linux_x86_64.whl

# Install server dependencies
$ENV_SERVER/bin/pip install omero-server
```

Download and unzip Omero.server:

```
cd /opt/omero/server
SERVER=https://downloads.openmicroscopy.org/omero/5.6/server-ice36.zip
wget -q $SERVER -O Omero.server-ice36.zip
unzip -q Omero.server*
```

Change the ownership of the Omero.server directory and create a symlink:

```
# change ownership of the folder
chown -R omero-server Omero.server-*
ln -s Omero.server-*/ Omero.server
```

Configuring Omero.server

The following steps are run as the omero-server system user. (su - omero-server)

The variable OMERODIR set in settings.env above **must** point to the location where Omero.server is installed. e.g. OMERODIR=/path_to_omero_server/OMero.server.

Note that this script requires the same environment variables that were set earlier in *settings.env*, so you may need to copy and/or source this file as the omero user.

Configure the database and the location of the data directory:

```
omero config set omero.data.dir "$OMERO_DATA_DIR"
omero config set omero.db.name "$OMERO_DB_NAME"
omero config set omero.db.user "$OMERO_DB_USER"
omero config set omero.db.pass "$OMERO_DB_PASS"
omero db script -f $OMERODIR/db.sql --password "$OMERO_ROOT_PASS"
psql -h localhost -U "$OMERO_DB_USER" "$OMERO_DB_NAME" < $OMERODIR/db.sql
```

Weaker ciphers like ADH are disabled by default in new versions of OpenSSL and TLS versions 1.0 and 1.1 have been dropped from JDK packages. In order to connect to an Omero.server using any Omero clients e.g. the Java Desktop client, the Omero.web client or the CLI and import data, you need to generate self-signed certificates after installing the *omero-certificates* package.

```
omero certificates
```

See also *Client Server SSL verification*.

Running OMERO.server

The following steps are run as the **omero-server** system user. (`su - omero-server`)

OMERO should now be set up. To start the server run:

```
omero admin start
```

Should you wish to start OMERO automatically, a *init.d* file could be created. An example `omero-server-init.d` is available.

Copy the *init.d* file and configure the service:

```
cp omero-server-init.d /etc/init.d/omero-server
chmod a+x /etc/init.d/omero-server

update-rc.d -f omero-server remove
update-rc.d -f omero-server defaults 98 02
```

You can then start up the service by running:

```
service omero-server start
```

Securing OMERO

The following steps are run as **root**.

If multiple users have access to the machine running OMERO you should restrict access to OMERO.server's configuration and runtime directories, and optionally the OMERO data directory:

```
chmod go-rwx $OMERODIR/etc $OMERODIR/var

# Optionally restrict access to the OMERO data directory
# chmod go-rwx "$OMERO_DATA_DIR"
```

OMERO.server installation on Ubuntu 22.04

This is an example walkthrough for installing OMERO on Ubuntu 22.04, using a dedicated local system user. You can use this as a guide for setting up your own test server. For production use you should also read the pages listed under [Optimizing Server Configuration](#). This guide will install Python 3.10. Since 5.6, a new `OMERODIR` variable is used, you should first unset `OMERO_HOME` (if set) before beginning the installation process.

This guide describes how to install using the **recommended** versions for Java, Ice, PostgreSQL. This should be read in conjunction with [Version requirements](#).

This guide does not describe how to install OMERO.web. To deploy OMERO.web, please read [OMERO.web installation on Ubuntu 22.04 and IcePy 3.6](#).

These instructions assume your Linux distribution is configured with a UTF-8 locale (this is normally the default).

For convenience in this walkthrough we will use the **omero-server** system user and the main OMERO configuration options have been defined as environment variables. When following this walkthrough you can either use your own values, or alternatively create `settings.env` for example under `/tmp` e.g. `/tmp/settings.env` containing the variables below and source it when required:

```
OMERO_DB_USER=db_user
OMERO_DB_PASS=db_password
OMERO_DB_NAME=omero_database
OMERO_ROOT_PASS=omero_root_password
OMERO_DATA_DIR=/OMERO

export OMEMO_DB_USER OMEMO_DB_PASS OMEMO_DB_NAME OMEMO_ROOT_PASS OMEMO_DATA_DIR

export PGPASSWORD="$OMERO_DB_PASS"

# Location of the OMEMO.server
export OMEMODIR=/opt/omero/server/OMERO.server

# Location of the virtual environment for omero-py
VENV_SERVER=/opt/omero/server/venv3

export ICE_HOME=/opt/ice-3.6.5

export PATH=$ICE_HOME/bin:$VENV_SERVER/bin:$PATH
```

Installing prerequisites

The following steps are run as root.

Install Java 11, Ice 3.6.5 and PostgreSQL 14:

To install Java 11 and other dependencies:

```
apt-get update

apt-get -y install unzip wget bc

# to be installed if daily cron tasks are configured
apt-get -y install cron

# install Java
apt-get update -q
apt-get install -y openjdk-11-jre

# install dependencies
```

To install Ice 3.6.5:

```
apt-get update && \
apt-get install -y -q \
db5.3-util \
bzip2 \
libdb++ \
libexpat1 \
libmcpp0 \
openssl \
```

(continues on next page)

(continued from previous page)

```
mcpp \
zlib1g

cd /tmp
wget -q https://github.com/glencoesoftware/zeroc-ice-ubuntu2204-x86_64/releases/download/
↪ 20221004/Ice-3.6.5-ubuntu2204-x86_64.tar.gz
tar xf Ice-3.6.5-ubuntu2204-x86_64.tar.gz
mv Ice-3.6.5 /opt/ice-3.6.5
echo /opt/ice-3.6.5/lib64 > /etc/ld.so.conf.d/ice-x86_64.conf
ldconfig
```

To make Ice available to all users and activate the virtual environment, set the following in `/etc/profile`:

```
# Environment file for OMERO

export ICE_HOME=/opt/ice-3.6.5
export PATH="$ICE_HOME/bin:$PATH"
#Remove commented out export below if Ice is not set globally accessible
#export LD_LIBRARY_PATH="$ICE_HOME/lib64:$ICE_HOME/lib:$LD_LIBRARY_PATH"
export SLICEPATH="$ICE_HOME/slice"
```

and add the virtual environment to `PATH`:

```
VENV_SERVER=/opt/omero/server/venv3

export ICE_HOME=/opt/ice-3.6.5

export PATH=$ICE_HOME/bin:$VENV_SERVER/bin:$PATH
```

To install PostgreSQL:

```
apt-get install -y gnupg
echo "deb http://apt.postgresql.org/pub/repos/apt jammy-pgdg main" > /etc/apt/sources.
↪ list.d/pgdg.list
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
apt-get update
apt-get -y install postgresql-15
service postgresql start
```

Create a local `omero-server` system user, and a directory for the OMERO repository:

```
useradd -mr omero-server
# Give a password to the omero user
# e.g. passwd omero-server
chmod a+X ~omero-server

mkdir -p "$OMERO_DATA_DIR"
chown omero-server "$OMERO_DATA_DIR"
```

Make the `settings.env` available to the `omero-server` system user by copying in to the user home directory. The file will need to be sourced each time you switch user. You could add `. ~/settings.env` to the `omero-server` system user bash profile.

Create a database user and initialize a new database for OMERO:

```
echo "CREATE USER $OMERO_DB_USER PASSWORD '$OMERO_DB_PASS'" | su - postgres -c psql
su - postgres -c "createdb -E UTF8 -O '$OMERO_DB_USER' '$OMERO_DB_NAME'"

psql -P pager=off -h localhost -U "$OMERO_DB_USER" -l
```

Installing OMERO.server

The following step is run as root.

We recommend to create a virtual environment and install the Ice Python binding and the dependencies required by the server using pip:

```
# Create a virtual env
python3 -mvenv $ENVV_SERVER

# Upgrade pip
$ENVV_SERVER/bin/pip install --upgrade pip

# Install the Ice Python binding
$ENVV_SERVER/bin/pip install https://github.com/glencoesoftware/zeroc-ice-py-ubuntu2204-
x86_64/releases/download/20221004/zeroc_ice-3.6.5-cp310-cp310-linux_x86_64.whl

# Install server dependencies
$ENVV_SERVER/bin/pip install omero-server
```

Download and unzip OMERO.server:

```
cd /opt/omero/server
SERVER=https://downloads.openmicroscopy.org/omero/5.6/server-ice36.zip
wget -q $SERVER -O OMERO.server-ice36.zip
unzip -q OMERO.server*
```

Change the ownership of the OMERO.server directory and create a symlink:

```
# change ownership of the folder
chown -R omero-server OMERO.server-*
ln -s OMERO.server-* / OMERO.server
```

Configuring OMERO.server

The following steps are run as the omero-server system user. (su - omero-server)

The variable OMERODIR set in settings.env above **must** point to the location where OMERO.server is installed. e.g. OMERODIR=/path_to_omero_server/OMERO.server.

Note that this script requires the same environment variables that were set earlier in settings.env, so you may need to copy and/or source this file as the omero user.

Configure the database and the location of the data directory:

```
omero config set omero.data.dir "$OMERO_DATA_DIR"
omero config set omero.db.name "$OMERO_DB_NAME"
```

(continues on next page)

(continued from previous page)

```
omero config set omero.db.user "$OMERO_DB_USER"
omero config set omero.db.pass "$OMERO_DB_PASS"
omero db script -f $OMERODIR/db.sql --password "$OMERO_ROOT_PASS"
psql -h localhost -U "$OMERO_DB_USER" "$OMERO_DB_NAME" < $OMERODIR/db.sql
```

Weaker ciphers like ADH are disabled by default in new versions of OpenSSL and TLS versions 1.0 and 1.1 have been dropped from JDK packages. In order to connect to an OMERO.server using any OMERO clients e.g. the Java Desktop client, the OMERO.web client or the CLI and import data, you need to generate self-signed certificates after installing the [omero-certificates](#) package.

```
omero certificates
```

See also [Client Server SSL verification](#).

Running OMERO.server

The following steps are run as the omero-server system user. (su - omero-server)

OMERO should now be set up. To start the server run:

```
omero admin start
```

Should you wish to start OMERO automatically, a *init.d* file could be created. An example *omero-server-init.d* is available.

Copy the *init.d* file and configure the service:

```
cp omero-server-init.d /etc/init.d/omero-server
chmod a+x /etc/init.d/omero-server

update-rc.d -f omero-server remove
update-rc.d -f omero-server defaults 98 02
```

You can then start up the service by running:

```
service omero-server start
```

Securing OMERO

The following steps are run as root.

If multiple users have access to the machine running OMERO you should restrict access to OMERO.server's configuration and runtime directories, and optionally the OMERO data directory:

```
chmod go-rwx $OMERODIR/etc $OMERODIR/var

# Optionally restrict access to the OMERO data directory
# chmod go-rwx "$OMERO_DATA_DIR"
```

OMERO.server installation on Debian 10

Warning: Debian 10 will reach End Of Life on June 30, 2024. We recommend to use Ubuntu 22.04 instead. Please read [OMERO.server installation on Ubuntu 22.04](#).

This is an example walkthrough for installing OMERO on Debian 10, using a dedicated local system user. You can use this as a guide for setting up your own test server. For production use you should also read the pages listed under [Optimizing Server Configuration](#). This guide will install Python 3.7. Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process.

This guide describes how to install using the **recommended** versions for Java, Ice, PostgreSQL. This should be read in conjunction with [Version requirements](#).

This guide does not describe how to install OMERO.web. To deploy OMERO.web, please read [OMERO.web installation on Debian 10 and IcePy 3.6](#).

These instructions assume your Linux distribution is configured with a UTF-8 locale (this is normally the default).

For convenience in this walkthrough we will use the **omero-server system user** and the main OMERO configuration options have been defined as environment variables. When following this walkthrough you can either use your own values, or alternatively create `settings.env` for example under `/tmp` e.g. `/tmp/settings.env` containing the variables below and source it when required:

```
OMERO_DB_USER=db_user
OMERO_DB_PASS=db_password
OMERO_DB_NAME=omero_database
OMERO_ROOT_PASS=omero_root_password
OMERO_DATA_DIR=/OMERO

export OMERO_DB_USER OMERO_DB_PASS OMERO_DB_NAME OMERO_ROOT_PASS OMERO_DATA_DIR

export PGPASSWORD="$OMERO_DB_PASS"

# Location of the OMERO.server
export OMERODIR=/opt/omero/server/OMERO.server

# Location of the virtual environment for omero-py
VENV_SERVER=/opt/omero/server/venv3

export ICE_HOME=/opt/ice-3.6.5

export PATH=$ICE_HOME/bin:$VENV_SERVER/bin:$PATH
```

Installing prerequisites

The following steps are run as root.

Install Java 11, Ice 3.6.5 and PostgreSQL 14:

To install Java 11 and other dependencies:

```
apt-get update

apt-get -y install unzip wget bc

# to be installed if daily cron tasks are configured
apt-get -y install cron

# install Java
apt-get -y install default-jre

# install dependencies

apt-get -y install \
    python3 \
    python3-venv
```

To install Ice 3.6.5:

```
apt-get update && \
apt-get install -y -q \
build-essential \
db5.3-util \
libbz2-dev \
libdb++-dev \
libdb-dev \
libexpat-dev \
libmcpp-dev \
libssl-dev \
mcpp \
zlib1g-dev

cd /tmp
wget -q https://github.com/ome/zeroc-ice-debian10/releases/download/0.1.0/ice-3.6.5-0.1.0-debian10-amd64.tar.gz
tar xf ice-3.6.5-0.1.0-debian10-amd64.tar.gz
mv ice-3.6.5-0.1.0 ice-3.6.5
mv ice-3.6.5 /opt
echo /opt/ice-3.6.5/lib/x86_64-linux-gnu > /etc/ld.so.conf.d/ice-x86_64.conf
ldconfig
```

To make Ice available to all users, set the environment using omero-ice36.env:

```
cat omero-ice36.env >> /etc/profile
```

To install PostgreSQL 14:

```
apt-get install -y postgresql-11
service postgresql start
```

Create a local omero-server system user, and a directory for the OMERO repository:

```
useradd -mr omero-server
# Give a password to the omero user
# e.g. passwd omero-server
chmod a+X ~omero-server

mkdir -p "$OMERO_DATA_DIR"
chown omero-server "$OMERO_DATA_DIR"
```

Make the `settings.env` available to the omero-server system user by copying in to the user home directory. The file will need to be sourced each time you switch user. You could add `. ~/settings.env` to the omero-server system user bash profile.

Create a database user and initialize a new database for OMERO:

```
echo "CREATE USER $OMERO_DB_USER PASSWORD '$OMERO_DB_PASS'" | su - postgres -c psql
su - postgres -c "createdb -E UTF8 -O '$OMERO_DB_USER' '$OMERO_DB_NAME'"

psql -P pager=off -h localhost -U "$OMERO_DB_USER" -l
```

Installing OMERO.server

The following step is run as root.

We recommend to create a virtual environment and install the Ice Python binding and the dependencies required by the server using pip:

```
# Create a virtual env
python3 -mvenv $ENVV_SERVER

# Upgrade pip
$ENVV_SERVER/bin/pip install --upgrade pip

# Install the Ice Python binding
$ENVV_SERVER/bin/pip install https://github.com/ome/zeroc-ice-debian10/releases/download/
0.1.0/zeroc_ice-3.6.5-cp37-cp37m-linux_x86_64.whl

# Install server dependencies
$ENVV_SERVER/bin/pip install omero-server
```

Download and unzip OMERO.server:

```
cd /opt/omero/server
SERVER=https://downloads.openmicroscopy.org/omero/5.6/server-ice36.zip
wget -q $SERVER -O OMERO.server-ice36.zip
unzip -q OMERO.server*
```

Change the ownership of the OMERO.server directory and create a symlink:

```
# change ownership of the folder
chown -R omero-server Omero.server-*
ln -s Omero.server-* / Omero.server
```

Configuring Omero.server

The following steps are run as the omero-server system user.

The variable OMERODIR set in `settings.env` above **must** point to the location where Omero.server is installed. e.g. `OMERODIR=/path_to_omero_server/Omero.server`.

Note that this script requires the same environment variables that were set earlier in `settings.env`, so you may need to copy and/or source this file as the omero user.

Configure the database and the location of the data directory:

```
omero config set omero.data.dir "$OMERO_DATA_DIR"
omero config set omero.db.name "$OMERO_DB_NAME"
omero config set omero.db.user "$OMERO_DB_USER"
omero config set omero.db.pass "$OMERO_DB_PASS"
omero db script -f $OMERODIR/db.sql --password "$OMERO_ROOT_PASS"
psql -h localhost -U "$OMERO_DB_USER" "$OMERO_DB_NAME" < $OMERODIR/db.sql
```

Weaker ciphers like ADH are disabled by default in new versions of OpenSSL and TLS versions 1.0 and 1.1 have been dropped from JDK packages. In order to connect to an Omero.server using any Omero clients e.g. the Java Desktop client, the Omero.web client or the CLI and import data, you need to generate self-signed certificates after installing the [omero-certificates](#) package.

```
omero certificates
```

See also [Client Server SSL verification](#).

Running Omero.server

The following steps are run as the omero-server system user.

OMERO should now be set up. To start the server run:

```
omero admin start
```

Should you wish to start Omero automatically, a `init.d` file could be created. An example `omero-server-init.d` is available.

Copy the `init.d` file and configure the service:

```
cp omero-server-init.d /etc/init.d/omero-server
chmod a+x /etc/init.d/omero-server

update-rc.d -f omero-server remove
update-rc.d -f omero-server defaults 98 02
```

You can then start up the service by running:

```
service omero-server start
```

Securing OMERO

The following steps are run as root.

If multiple users have access to the machine running OMERO you should restrict access to OMERO.server's configuration and runtime directories, and optionally the OMERO data directory:

```
chmod go-rwx $OMERODIR/etc $OMERODIR/var

# Optionally restrict access to the OMERO data directory
# chmod go-rwx "$OMERO_DATA_DIR"
```

OMERO.server installation on OS X with Homebrew

Overview

This walkthrough demonstrates how to install OMERO on a clean Mac OS X system (10.9 or later). Dependencies are installed with Homebrew. The OMERO.server can be downloaded as a pre-built zip, or built from the source code. It is aimed at **developers** since typically MacOS X is not suited for serious server deployment.

Prerequisites

Xcode

Homebrew requires the latest version of Xcode. Install **Xcode** and the Command Line Tools for Xcode from the App Store. If you have already installed it, make sure all the latest updates are installed.

Homebrew

Homebrew will install all packages under `/usr/local`. See also: Installation instructions on the [Homebrew wiki](#).

Install Homebrew using the following command in terminal:

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/
↪install)"
```

Java

Java may be installed using OpenJDK from [AdoptOpenJDK](#). See [Version requirements](#) for supported versions.

After installing JDK, check your installation works by running:

```
$ java --version
openjdk 11.0.5 2019-10-15
OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.5+10)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.5+10, mixed mode)

$ javac -version
javac 11.0.5
```

OS X Basics

In order to develop on OMERO, we recommend you ensure you have your Mac setup for development. The first step to achieving this is to create a `.bash_profile` file in the root directory of your user folder.

To create a `.bash_profile` from terminal, if one does not already exist:

```
$ touch ~/.bash_profile
```

To open your `.bash_profile` in a text editor, such as the built-in TextEdit app, use:

```
$ open -a TextEdit.app ~/.bash_profile
```

Note: If you want changes to your `.bash_profile` to take effect without restarting OS X, run:

```
$ source ~/.bash_profile
```

Requirements

1. Open a command-line terminal and install git if not already present:

```
$ brew install git
```

2. Install PostgreSQL database server:

```
$ brew install postgresql
```

To ensure PostgreSQL uses UTF-8 encoding, open your bash profile and add the following environment variables:

```
export LANG=en_US.UTF-8
export LANGUAGE=en_US:en
```

3. Install NGINX:

```
$ brew install nginx
```

4. OMERO depends on Ice 3.6 and unfortunately does not run with the Ice version 3.7 or higher. To obtain Ice 3.6, we need to add a *tap* to Homebrew:

```
$ brew tap zeroc-ice/tap
$ brew install zeroc-ice/tap/ice36
```

Note: If you already have a version of Ice that is not 3.6 installed, you can instruct Homebrew to *unlink* it using ``$ brew unlink ice``. You can then instruct Homebrew to link to Ice 3.6 using ``$ brew link ice@36``

Python

For developing with OMERO, or Python in general, we recommend the use of Virtualenv. Virtualenv allows development of Python applications without having to worry about clashing third-party packages for different Python projects.

We will create 2 virtual environments below, one for `omero-py` and another for `omero-web` (which also includes `omero-py`). This allows more flexibility, but you can use just the `omero-web` virtual environment for everything if you wish.

You can create virtual environments using either `conda` (preferred) OR `venv`.

Using conda (preferred)

1. Install Conda. See [miniconda](#) for more details.
2. Create virtual environments named `omeropy`:

```
$ conda create -n omeropy -c conda-forge python=3.8 zeroc-ice omero-py
```

3. Create virtual environments named `omeroweb`, activate it and install dependencies:

```
$ conda create -n omeroweb -c conda-forge python=3.8 zeroc-ice omero-py
$ conda activate omeroweb
$ pip install "omero-web>=\ |version_web|"
```

4. Activate the virtual environments:

```
$ conda activate omeropy
```

5. You can now use the `omero` command. You will also need to ensure you are in the appropriate environment when you install additional modules:

```
$ omero -h

# Additional modules. For example:
$ pip install omero-metadata
```

Now go to the [OMERO installation](#) section below.

OR using venv

1. install Python provided by Homebrew:

```
$ brew install python
```

Follow the instructions from the brew Python install and set your system to use the Homebrew version of Python rather than the Python shipped with OS X. Typically:

```
$ brew link python
```

2. Check that Python is working and is version 3.7.x:

```
$ which python3
/usr/local/bin/python3

$ python3 --version
Python 3.7.4
```

3. Create a virtual environments for omero-py and/or omero-web using Python 3:

```
$ python3 -mvenv ~/Virtual/omero-py
$ python3 -mvenv ~/Virtual/omero-web
```

4. Activate the Virtualenv environment(s) and install modules:

```
$ source ~/Virtual/omero-py/bin/activate
$ pip install "omero-py>=\ |version_py|"

# In a different terminal:
$ source ~/Virtual/omero-web/bin/activate
$ pip install "omero-web>=\ |version_web|"
```

5. You can now use the omero command in either virtual environment. You will also need to ensure you are in the appropriate environment when you install additional modules:

```
$ omero -h

# Additional modules. For example:
$ pip install omero-metadata
```

OMERO installation

Pre-built server

1. Using the command-line terminal, prepare a place for your OMERO server to be downloaded to.
2. Find the current OMERO.server zip from the [downloads page](#).
3. Download and extract the OMERO.server-x.x.x-ice36-bxx.zip.

Locally built server

1. Clone the source code from the project's GitHub account to build locally:

```
$ git clone --recursive https://github.com/ome/openmicroscopy
```

2. Navigate terminal into the `openmicroscopy` that was just created by performing the previous step:

```
$ cd openmicroscopy
```

3. Execute the build script (*this will take a few minutes, depending on how fast your Mac is*)

```
$ ./build.py
```

4. Once the build completes, the OMERO server build output will be located in `openmicroscopy/dist`.

See also:

Installing OMERO from source

Developer documentation page on how to check out to source code

Build System

Developer documentation page on how to build the OMERO.server

OMERO configuration

1. Open your `.bash_profile` in a text editor, such as the built-in TextEdit app:

```
$ open -a TextEdit.app ~/.bash_profile
```

2. Add an environment variable `OMERODIR` to the `.bash_profile` which points to the location of the OMERO executable:

```
# Pre-built server...
export OMERODIR=/path/to/OMERO.server-x.x.x-ice36-bxx
# ...OR locally built server
export OMERODIR=/path/to/openmicroscopy/dist
```

3. Using the command-line terminal, reload your `.bash_profile` using:

```
$ source ~/.bash_profile
```

Database

1. From a fresh command-line terminal, start the database server:

```
$ pg_ctl -D /usr/local/var/postgres -l /usr/local/var/postgres/server.log -w start
```

2. To use OMERO, we need to first set up PostgreSQL. Open a command-line terminal and run the following commands to create a user called `db_user` and a database called `omero_database`:

```
$ createuser -w -D -R -S db_user
$ createdb -E UTF8 -O db_user omero_database
```

3. Activate the omeropy env:

```
$ conda activate omeropy
# OR
$ source ~/Virtual/omeropy/bin/activate
```

4. Now set the OMERO configuration:

```
$ omero config set omero.db.name omero_database
$ omero config set omero.db.user db_user
$ omero config set omero.db.pass db_password
```

5. Create and run script to initialize the OMERO database:

```
$ omero db script --password omero -f - | psql -h localhost -U db_user omero_
↪database
```

Note: (Optional) To make life easier, you can add an `alias` to your `.bash_profile` to start and stop the Postgres service:

```
alias startPg='pg_ctl -D /usr/local/var/postgres -l /usr/local/var/postgres/server.log -
↪w start'
alias stopPg='pg_ctl -D /usr/local/var/postgres -l /usr/local/var/postgres/server.log -w
↪stop'

Reload :file:`.bash_profile` in OS X::

$ source ~/.bash_profile
```

Binary Repository

1. Create directory for OMERO to store its data:

```
$ mkdir /OMERO
$ omero config set omero.data.dir /OMERO
```

OMERO.web

1. Activate the omeroweb env:

```
$ conda activate omeroweb
# OR
$ source ~/Virtual/omeroweb/bin/activate
```

2. Basic setup for OMERO using NGINX:

```
$ mv /usr/local/etc/nginx/nginx.conf /usr/local/etc/nginx/nginx.conf.orig
$ omero web config nginx-development > /usr/local/etc/nginx/nginx.conf
$ nginx -t
$ nginx
```

Note: The internal Django webserver can be used for evaluation and development. In this case please follow the instructions under [OMERO.web installation for developers](#).

Startup and shutdown

Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process. The variable OMERODIR **must** point to the location where OMERO.server is installed. e.g. OMERODIR=/path_to_omero_server/OMERO.server.

If necessary start PostgreSQL database server:

```
$ pg_ctl -D /usr/local/var/postgres -l /usr/local/var/postgres/server.log -w start
```

Activate the omeropy env and start OMERO:

```
$ conda activate omeropy
# OR
$ source ~/Virtual/omeropy/bin/activate
$ omero admin start
```

Activate the omeroweb env and start OMERO.web:

```
$ conda activate omeroweb
# OR
$ source ~/Virtual/omeroweb/bin/activate
$ omero web start
```

Now connect to your OMERO.server using OMERO.insight or OMERO.web with the following credentials:

```
U: root
P: omero
```

Activate the omeroweb env as above, and stop OMERO.web:

```
$ omero web stop
```

Activate the omeropy env as above and stop OMERO:

```
$ omero admin stop
```

Web configuration and maintenance

For more configuration options and maintenance advice for OMERO.web see [OMERO.web installation and maintenance](#).

Common issues

General considerations

If you run into problems with Homebrew, you can always run:

```
$ brew update
$ brew doctor
```

Also, please check the Homebrew [Bug Fixing Checklist](#).

Below is a non-exhaustive list of errors/warnings specific to the OMERO installation. Some if not all of them could possibly be avoided by removing any previous OMERO installation artifacts from your system.

Database

Check to make sure the database has been created and 'UTF8' encoding is used

```
$ psql -h localhost -U db_user -l
```

This command should give similar output to the following:

List of databases					
Name	Owner	Encoding	Collation	Ctype	Access privileges
omero_database	db_user	UTF8	en_GB.UTF-8	en_GB.UTF-8	
postgres	ome	UTF8	en_GB.UTF-8	en_GB.UTF-8	
template0	ome	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/ome +
template1	ome	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/ome +
					ome=CTc/ome
(4 rows)					

PostgreSQL

If you encounter this error during installation of PostgreSQL:

```
Error: You must ``brew link ossp-uuid' before postgresql can be installed
```

try:

```
$ brew cleanup
$ brew link ossp-uuid
```

For recent versions of OS X (10.10 and above) some directories may be missing, preventing PostgreSQL from starting up. In that case, it should be sufficient to reinitialize a PostgreSQL database cluster as:

```
$ rm -rf /usr/local/var/postgres
$ initdb -E UTF8 /usr/local/var/postgres
```

See also:

<https://stackoverflow.com/questions/25970132/pg-tblspc-missing-after-installation-of-latest-version-of-os-x-yosemite-or-el>

szip

If you encounter an MD5 mismatch error similar to this:

```
==> Installing hdf5 dependency: szip
==> Downloading http://www.hdfgroup.org/ftp/lib-external/szip/2.1/src/szip-2.1.tar.gz
Already downloaded: /Library/Caches/Homebrew/szip-2.1.tar.gz
Error: MD5 mismatch
Expected: 902f831bcefb69c6b635374424acbead
Got: 0d6a55bb7787f9ff8b9d608f23ef5be0
Archive: /Library/Caches/Homebrew/szip-2.1.tar.gz
(To retry an incomplete download, remove the file above.)
```

then manually remove the archived version located under `/Library/Caches/Homebrew`, since the maintainer may have updated the file.

numexpr (and other Python packages)

If you encounter an issue related to numexpr complaining about NumPy having too low a version number, verify that you have not previously installed any Python packages using **pip**. In the case where **pip** has been installed before Homebrew, uninstall it:

```
$ sudo pip uninstall pip
```

and then try running `python_deps.sh` again. That should install **pip** via Homebrew and put the Python packages in correct locations.

Prerequisites

Installation will require:

- a clean, minimal operating system installation
- a “root” level account for which you know the password

Note: If you are unsure of what it means to have a “root” level account, or if you are generally having issues with the various users/passwords described in this install guide, please see *[Which user account and password do I use where?](#)*.

The installation and configuration of the prerequisite applications are mostly outside the scope of this document. For Linux distributions, use of the default package manager is recommended. For MacOS X, Homebrew is recommended. This guide provides the package names to install for a number of contemporary systems. However, the names and versions provided vary between releases. Please do check for similar packages if the one documented here is not available for your system as it may be provided under an alternative name. “Debian” refers to Debian and derivative distributions such as Ubuntu. “RedHat” refers to RedHat and related distributions such as CentOS, Fedora and Scientific Linux.

- For Ubuntu you need to enable the **universe** repository. This should be enabled by default. If not enabled, it may be enabled by editing `/etc/apt/sources.list` directly, in which case the entries may already exist but

are commented out, or by using Synaptic (10.04 and 10.10) or Ubuntu Software Center (11.04 onwards). Update your package lists to ensure that you get the latest packages:

```
$ sudo apt-get update
```

Install packages by running:

```
$ sudo apt-get install package
```

where *package* is the package name to install.

The following subsections cover the details for each package, in the order recommended for installation.

Java SE Runtime Environment (JRE)

If possible, install one of the following packages:

System	Package
Debian	openjdk-11-jre
Homebrew	N/A (install Oracle Java)
RedHat	java-11-openjdk

OMERO works with the OpenJDK JRE provided by most systems, or with Oracle Java. Version 8 or later is required. Version 11 is recommended.

Your system may already provide a suitable JRE, in which case no extra steps are necessary. Linux distributions usually provide OpenJDK, and older MacOS X versions have Java installed by default. Oracle Java is no longer provided by BSD or Linux distributions for licensing reasons. If your system does not have Java available, for example on newer MacOS X versions, or the provided version is too old, Oracle Java may be downloaded from the [Oracle website](#).

Warning: Security

Installing Oracle Java outside the system's package manager will leave your system without regular distribution-supplied security updates, and so is not recommended.

To check which version of Java is currently available:

```
$ which java
/usr/bin/java
$ java -version
openjdk version "11.0.5" 2019-10-15
OpenJDK Runtime Environment (build 11.0.5+10-post-Ubuntu-0ubuntu1.118.04)
OpenJDK 64-Bit Server VM (build 11.0.5+10-post-Ubuntu-0ubuntu1.118.04, mixed mode,
↳ sharing)
```

Python 3

Check you have Python (and check its version) by running:

```
$ python3 --version
Python 3.6.4
```

If possible, install the following packages:

System	Package
Debian	python3
Homebrew	python3
RedHat	python3

Ice

The Ice version may vary, depending upon the distribution version you are using. The Ice versions in currently supported versions of Debian and [Ubuntu](#) are shown in the *Ice* of the [Version requirements](#) page.

Using version 3.6 of Ice is required. If your package manager provides Ice packages, using these is recommended where possible. Distribution-provided packages often have additional bugfixes which are not present in the upstream releases.

If needed, source and binary packages are available from [ZeroC](#). The latest release is available from the [ZeroC website](#).

Note: [ZeroC](#) Ice can always be built from source code for specific platforms if a binary package is not available.

Note: With Ice 3.6, the Python bindings are provided separately. If your package manager does not provide Ice python packages, run `pip install zeroc-ice` to install the Ice Python bindings. See [Using the Python Distribution](#) for further details.

OMERO.scripts

If you wish to run the “Movie Maker” script, please install **mencoder**.

System	Package
Debian	mencoder
Homebrew	mplayer
RedHat	mencoder

Installation

Once the above prerequisites have been downloaded, installed and configured appropriately, the OMERO server itself may be installed. You may wish to create a user account solely for the purpose of running the server, and switch to this user for the next steps.

Server directory

Firstly, a directory needs to be created to contain the server. In this case `~/omero` is used as an example:

```
$ mkdir -p ~/omero
```

Next, change into this directory:

```
$ cd ~/omero
```

OMERO.server

The release `OMERO.server.zip` is available from the [OMERO downloads](#) page. Download the version matching the version of Ice installed on your system before continuing.

Installing a development version from source is also possible. See the [Installing OMERO from source](#) section for further details. This is not recommended unless you have a specific reason *not* to use a release version.

Once you have obtained the OMERO.server zip archive matching the version of Ice installed, unpack it:

```
$ unzip OMERO.server-5.6.10-ice36.zip
```

If your system does not provide an **unzip** command by default, install one of the following:

System	Package
Debian	unzip
Homebrew	unzip
RedHat	unzip

Optionally, give your OMERO software install a short name to save some typing later, to reflect what you set `OMERO_PREFIX` to in the [Environment variables](#) section, below:

```
$ ln -s OMERO.server-5.6.10-ice36 OMERO.server
```

This will also ease installation of newer versions of the server at a later date, by simply updating the link.

Note: Weaker ciphers like ADH are disabled by default in new versions of OpenSSL and TLS versions 1.0 and 1.1 have been dropped from JDK packages. In order to connect to an OMERO.server using any OMERO clients e.g. the Java Desktop client, the OMERO.web client or the CLI and import data, you need to generate self-signed certificates after installing the [omero-certificates](#) package.

Environment variables

If using distribution-provided packages such as Debian or RPM packages, or via the homebrew or macports package manager, it should not be necessary to set any environment variables. However, if using third-party packages for any required components, several variables may require setting in order for them to function correctly.

Please note that the precise details of these environment variables can change as new versions of software are released.

There are several methods for setting environment variables; which is most appropriate will depend upon how the OMERO server is started. Options include:

/etc/security/pam_env.conf

Global environment set at login by PAM

/etc/profile or **/etc/profile.d/omero**

Global Bourne shell defaults (also used by derived shells such as **bash** and **zsh**)

~/ .profile

User's Bourne shell defaults (also used by derived shells)

/etc/bash.bashrc

Global **bash** defaults

~/ .bashrc, **~/ .bash_profile** or **~/ .bash_login**

User's **bash** configuration.

If OMERO is started as a service using an init script, a global setting should be preferred. If being started by hand using a particular user, a user-specific configuration file may be more appropriate.

The following environment variables may be configured:

LD_LIBRARY_PATH (Linux) or DYLD_LIBRARY_PATH (MacOS X)

The Ice and PostgreSQL libraries must be on the library search path. If using the packages provided by your distribution, this will already be the case. If using third-party binary distributions the **lib** (or **lib64** if present and using a 64-bit system) directory for each will require adding to the library search path.

OMERO_PREFIX

This is not strictly required, but may be set for convenience to point to the OMERO server installation, and is used in this documentation as a shorthand for the installation path.

OMERO_TMPDIR

Directory used for temporary files. If the home directory of the user running the OMERO server is located on a slow filesystem, such as NFS, this may be used to store the temporary files on fast local storage.

PATH

The search path must include the programs **java**, **python**, **icegridnode** and PostgreSQL commands such as **psql**. If using the packages provided by your distribution, this will already be the case. If using third-party binary distributions such as the ZeroC Ice package, Oracle Java, or PostgreSQL, the **bin** directory for each must be added to the path. The OMERO **bin** directory may also be added to the search path (**\$OMERO_PREFIX/bin** if **OMERO_PREFIX** has been set).

PYTHONPATH

The Ice **python** directory must be made available to python. If using the Ice packages provided by your distribution, this will already be the case. If using the ZeroC ice package, add the **python** directory to the python path. For Ice 3.6, this should never be required.

OMERODIR

The path to the OMERO.server. This is a requirement for all CLI plugins using the Java server components (*admin*, *import*, *config*, *db*...).

After making any needed changes, either source the corresponding file or log back in for them to take effect. Run **env** to check them.

Creating a database

On most systems, a “postgres” user will be created which has admin privileges, while the UNIX root user itself does *not* have admin privileges. Therefore it is necessary to either become the **postgres** user, or use **sudo** as shown below.

For the purposes of this guide, the following dummy data is used:

```
Username: db_user
Password: db_password
Database: omero_database
```

Warning: Security

These dummy values are examples only and should **not** be used. For a live or public server install these values should be altered to reflect your security requirements—i.e. use your own choice of username and password instead. These should **not** be the same username and/or password as your Linux/Mac root user!

You should also consider restricting access to your server machine, but that is outside the scope of this document.

- Create a non-superuser database user and record the name and password used. You will need to configure OMERO to use this username and password later on.:

```
$ sudo -u postgres createuser -P -D -R -S db_user
Enter password for new role:      # db_password
Enter it again:                  # db_password
```

- Create a database for OMERO to reside in:

```
$ sudo -u postgres createdb -E UTF8 -O db_user omero_database
```

- Check to make sure the database has been created, you have PostgreSQL client authentication correctly set up and the database is owned by the **db_user** user.

```
$ psql -h localhost -U db_user -l
Password for user db_user:
      List of databases
  Name          | Owner   | Encoding
-----+-----+-----
 omero_database | db_user | UTF8
 postgres      | postgres | UTF8
 template0     | postgres | UTF8
 template1     | postgres | UTF8
(4 rows)
```

If you have problems, especially with the last step, take a look at [OMERO.server and PostgreSQL](#) since the authentication mechanism is probably not properly configured.

Location for the your OMERO binary repository

- Create a directory for the OMERO binary data repository. `/OMERO` is the default location and should be used unless you explicitly have a reason not to and know what you are doing.
- This is *not* where you want the OMERO application to be installed, it is a *separate* directory which the OMERO.server will use to store binary data.
- You can read more about the [OMERO binary repository](#).

```
$ sudo mkdir /OMERO
```

- Change the ownership of the directory. `/OMERO` **must** either be owned by the user starting the server (it is currently owned by the system root) or that user **must** have permission to write to the directory. You can find out your username and edit the correct permissions as follows:

```
$ whoami
omero
$ sudo chown -R omero /OMERO
```

Configuration

- You can view a parsed version of the configuration properties under [Configuration properties glossary](#) or parse it yourself with `omero config parse`.
- Change any settings that are necessary using `omero config`, including the name and/or password for the 'db_user' database user you chose above or the database name if it is not "omero_database". (Quotes are only necessary if the value could be misinterpreted by the shell. See [Forum post](#))

```
$ omero config set omero.db.name 'omero_database'
$ omero config set omero.db.user 'db_user'
$ omero config set omero.db.pass 'db_password'
```

You can also check the values that have been set using:

```
$ omero config get
```

- If you have chosen a non-standard [OMERO binary repository](#) location above, be sure to configure the `omero.data.dir` property. For example, to use `/srv/omero`:

```
$ omero config set omero.data.dir /srv/omero
```

- Create the OMERO database initialization script. You will need to provide a password for the newly created OMERO root user, either by using the `--password` argument or by entering it when prompted. Note that this password is for the root user of the **OMERO.server**, and is not related to the root system user or a PostgreSQL user role.

```
$ omero db script --password omero_root_password
```

```
Using OMERO5.4 for version
Using 0 for patch
Using password from commandline
Saving to /home/omero/OMERO5.4__0.sql
```

Warning: Security

For illustrative purposes, the default password for the OMERO root user is shown as `omero_root_password`. However, you should **not** use this default values for your installation, but use your own choice of password instead. This should **not** be the same password as your Linux/Mac root user or the database user!

- Initialize your database with the script.

```
$ psql -h localhost -U db_user omero_database < OMERO5.4__0.sql
```

At this point you should see some output from PostgreSQL as it installs the schema for new OMERO database.

- Before starting the OMERO.server, run the OMERO diagnostics script to check that all of the settings are correct, e.g.

```
$ omero admin diagnostics
```

- You can now start the server using:

```
$ omero admin start
Creating var/master
Initializing var/log
Creating var/registry
No descriptor given. Using etc/grid/default.xml
```

- If multiple users have access to the system running OMERO you should restrict access to the `OMERO.server/etc` and `OMERO.server/var` directories, for example by changing the permissions on them:

```
$ chmod 700 ~/omero/OMERO.server/etc ~/omero/OMERO.server/var
```

You should also consider restricting access to the OMERO data repository. The required permissions will depend on whether you are using *Advanced import scenarios*.

- Test that you can log in as “root”, either with the OMERO.insight client or on the command-line:

```
$ omero login
Server: [localhost]
Username: [root]
Password:          # omero_root_password
```

You will be prompted for an OMERO username and password. Use the username and password set when running **omero db script**.

- If your users are going to be importing many files in one go, for example multiple plates, you should make sure you set the maximum number of open files to a sensible level (i.e. at least 8K for production systems, 16K for bigger machines). See *Too many open files* for more information.

JVM memory settings

The OMERO server starts a number of Java services. Memory settings for these are calculated on a system-by-system basis. An attempt has been made to have usable settings out of the box, but if you can afford to provide OMERO with more memory, it will certainly improve your overall performance. See [Memory configuration](#) on how to tune the JVM.

Enabling movie creation from OMERO

OMERO has a facility to create AVI/MPEG Movies from images. The page [OMERO.movie](#) details how to enable it.

Post-installation items

Backup

One of your first steps after putting your OMERO server into production should be deciding on when and how you are going to [backup your database and binary data](#). Please do not omit this step.

Security

It is also now recommended that you read the [Server security and firewalls](#) page to get a good idea as to what you need to do to get OMERO clients speaking to your newly installed OMERO.server in accordance with your institution or company's security policy.

Advanced configuration

Once you have the base server running, you may want to try enabling some of the advanced features such as [OMERO.dropbox](#) or [LDAP authentication](#). If you have **Flex data**, you may want to watch the [HCS configuration screencast](#). See [Configuration properties glossary](#) on how to get the most out of your server.

Troubleshooting

My OMERO install doesn't work! What do I do now? Examine the [Troubleshooting OMERO](#) page and if all else fails post a message to the [forum](#) mentioned on the [Community support](#) page. Especially the [Server fails to start](#) and [Remote clients cannot connect to OMERO installation](#) sections are a good starting point.

OMERO diagnostics

If you want help with your server installation, please include the output of the diagnostics command:

```
$ omero admin diagnostics
```

```
=====
OMERO Diagnostics 5.6.10
=====
```

```
Commands:  java -version           11.0.5    (/usr/bin/java)
Commands:  python -V              3.6.9    (/opt/omero/server/venv3/bin/python)
Commands:  icegridnode --version  3.6.5    (/usr/bin/icegridnode)
Commands:  icegridadmin --version 3.6.5    (/usr/bin/icegridadmin)
```

```

Commands:  psql --version          11.6      (/usr/bin/psql)
Commands:  openssl version        1.1.111  (/usr/bin/openssl)

Server:    icegridnode            running
Server:    Blitz-0                active (pid = 30324, enabled)
Server:    DropBox                active (pid = 30343, enabled)
Server:    FileServer             active (pid = 30345, enabled)
Server:    Indexer-0              active (pid = 30348, enabled)
Server:    MonitorServer          active (pid = 30351, enabled)
Server:    OMERO.Glacier2         active (pid = 30353, enabled)
Server:    OMERO.IceStorm         active (pid = 30376, enabled)
Server:    PixelData-0            active (pid = 30393, enabled)
Server:    Processor-0            active (pid = 30394, enabled)
Server:    Tables-0               inactive (disabled)
Server:    TestDropBox            inactive (enabled)

OMERO:     SSL port                4064
OMERO:     TCP port                4063

Log dir:   /opt/omero/server/OMERO.server/var/log exists

Log files: Blitz-0.log             22.8 KB      errors=0      warnings=9
Log files: DropBox.log            1.3 KB      errors=0      warnings=1
Log files: FileServer.log         114 B
Log files: Indexer-0.log          1.3 KB      errors=0      warnings=5
Log files: MonitorServer.log      882 B
Log files: PixelData-0.log        1.8 KB      errors=0      warnings=4
Log files: Processor-0.log        592 B      errors=0      warnings=1
Log files: Tables-0.log           841 B
Log files: TestDropBox.log        n/a
Log files: master.err             34.4 KB      errors=2      warnings=0
Log files: master.out             empty
Log files: Total size             0.06 MB

Environment:OMERODIR=/opt/omero/server/OMERO.server
Environment:OMERO_HOME=(unset)
Environment:OMERO_NODE=(unset)
Environment:OMERO_MASTER=(unset)
Environment:OMERO_TEMPDIR=(unset)
Environment:PATH=/opt/omero/server/venv3/bin:/usr/local/bin:/usr/bin:/bin
Environment:ICE_HOME=(unset)
Environment:LD_LIBRARY_PATH=(unset)
Environment:DYLD_LIBRARY_PATH=(unset)

OMERO SSL port:4064
OMERO TCP port:4063
OMERO data dir: '/OMERO'          Exists? True   Is writable? True
OMERO temp dir: '/home/omero-server/tmp'  Exists? True   Is writable? True
↪(Size: 0)

JVM settings: Blitz-${index}      -Xmx621m -XX:MaxPermSize=512m
↪-XX:+IgnoreUnrecognizedVMOptions
JVM settings: Indexer-${index}    -Xmx414m -XX:MaxPermSize=512m
↪-XX:+IgnoreUnrecognizedVMOptions

```

```
JVM settings: PixelData- $\{\text{index}\}$  -Xmx621m -XX:MaxPermSize=512m
↪-XX:+IgnoreUnrecognizedVMOptions
JVM settings: Repository- $\{\text{index}\}$  -Xmx414m -XX:MaxPermSize=512m
↪-XX:+IgnoreUnrecognizedVMOptions
```

Update notification

Your OMERO.server installation will check for updates each time it is started from the *Open Microscopy Environment* update server. If you wish to disable this functionality you should do so now as outlined on the [OMERO upgrade checks](#) page.

2.2.2 OMERO.web installation and maintenance

OMERO.web is a Python 3 client of the OMERO platform that provides a web-based UI and JSON API. This section provides links to detailed step-by-step walkthroughs describing how to install, customize, maintain and run OMERO.web for several systems. OMERO.web is installed **separately** from the OMERO.server.

OMERO.web can be deployed with:

- **WSGI** using a WSGI capable web server such as **NGINX** and **Gunicorn**
- the built-in Django lightweight development server. This type of deployment should only be used for **testing** purpose only; see the [OMERO.web installation for developers](#) page.

If you need help configuring your firewall rules, see [Server security and firewalls](#) for more details.

Depending upon which platform you are using, you may find a more specific walkthrough listed below. The guides use the example of deploying OMERO.web with **NGINX** and **Gunicorn**. OMERO can automatically generate a configuration file for your webserver. The location of the file will depend on your system, please refer to your webserver's manual. See in the section *Customizing your OMERO.web installation* in the various walkthroughs for more options.

Configuration

You will find in the various guides how to create the NGINX OMERO configuration file and the configuration steps for the NGINX and Gunicorn. Advanced Gunicorn setups are also described to enable the download of binary data and to handle multiple clients on a single worker thread switching context as necessary while streaming binary data from OMERO.server. Depending on the traffic and scale of the repository you should configure connections and speed limits on your server to avoid blocking resources.

Walkthroughs

Recommended:

[OMERO.web installation on Rocky Linux 9 and IcePy 3.6](#)

Instructions for installing OMERO.web from scratch on RHEL 9/Rocky Linux 9 with Ice 3.6.

[OMERO.web installation on Ubuntu 22.04 and IcePy 3.6](#)

Instructions for installing OMERO.web from scratch on Ubuntu 22.04 with Ice 3.6.

Supported:

[OMERO.web installation on CentOS 7 and IcePy 3.6](#)

Instructions for installing OMERO.web from scratch on CentOS 7 with Ice 3.6.

[OMERO.web installation on Ubuntu 20.04 and IcePy 3.6](#)

Instructions for installing OMERO.web from scratch on Ubuntu 20.04 with Ice 3.6.

OMERO.web installation on CentOS 7 and IcePy 3.6

Please first read *OMERO.server installation on CentOS 7*.

This is an example walkthrough for installing OMERO.web in a **virtual environment** using a dedicated system user. Installing OMERO.web in a virtual environment is the preferred way. For convenience in this walkthrough, we will use the **omero-web system user** and define the main OMERO.web configuration options as environment variables. Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process. By default, Python 3.6 is installed.

The following steps are run as root.

If required, first create a local system user omero-web and create directory:

```
useradd -m omero-web

mkdir -p /opt/omero/web/omero-web/etc/grid
chown -R omero-web /opt/omero/web/omero-web
```

Installing prerequisites

The following steps are run as root.

Install dependencies:

```
yum -y install epel-release

yum -y install unzip

yum -y install python3

yum -y install nginx
```

Optional: if you wish to use the Redis cache, install Redis:

```
yum -y install redis

systemctl enable redis.service

systemctl start redis.service
```

Creating a virtual environment

The following steps are run as root.

Create the virtual environment. This is the recommended way to install OMERO.web:

```
python3 -mvenv /opt/omero/web/venv3
```

Install ZeroC IcePy 3.6:

```
/opt/omero/web/venv3/bin/pip install https://github.com/ome/zeroc-ice-py-centos7/
↳ releases/download/0.2.1/zeroc_ice-3.6.5-cp36-cp36m-linux_x86_64.whl
```

Upgrade pip and install OMERO.web:

```
/opt/omero/web/venv3/bin/pip install --upgrade pip
/opt/omero/web/venv3/bin/pip install omero-web
```

Installing OMERO.web apps

A number of apps are available to add functionality to OMERO.web, such as [OMERO.figure](#) and [OMERO.iviewer](#). See the main website for a [list of released apps](#). These apps are optional and can be installed, as the **root user**, via **pip** to your OMERO.web virtual environment and configured as the **omero-web system user**, at any time.

Configuring OMERO.web

The following steps are run as the **omero-web system user**.

For convenience the main OMERO.web configuration options have been defined as environment variables. You can either use your own values, or alternatively use the following ones:

```
export WEBSSESSION=True
export OMERODIR=/opt/omero/web/omero-web
```

Configure OMERO.web and create the NGINX OMERO configuration file to be included in a system-wide NGINX configuration by redirecting the output of the command `omero web config nginx` below into a file. If an attempt is made to access OMERO.web whilst it is not running, the generated NGINX configuration file will automatically display a maintenance page:

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

omero web config nginx --http "${WEBPORT}" --servername "${WEBSERVER_NAME}" > /opt/omero/
↪web/omero-web/nginx.conf.tmp
```

OMERO.web offers a number of configuration options. The configuration changes **will not be applied** until Gunicorn is restarted using `omero web restart`. The Gunicorn workers are managed **separately** from other OMERO processes. You can check their status or stop them using `omero web status` or `omero web stop`.

- Session engine:
- OMERO.web offers alternative session backends to automatically delete stale data using the cache session store backend, see [Django cached session documentation](#) for more details.
- [Redis](#) requires [django-redis](#) in order to be used with OMERO.web. We assume that Redis has already been installed. To configure the cache, run:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.
↪cache.
RedisCache", "LOCATION": "redis://127.0.0.1:6379/0"}}'
```

- After installing all the cache prerequisites set the following:

```
omero config set omero.web.session_engine django.contrib.sessions.backends.
↪cache
```

- Use a prefix:

By default OMERO.web expects to be run from the root URL of the webserver. This can be changed by setting `omero.web.prefix` and `omero.web.static_url`. For example, to make OMERO.web appear at `http://example.org/omero/`:

```
omero config set omero.web.prefix '/omero'
omero config set omero.web.static_url '/omero/static/'
```

and regenerate your webserver configuration.

All configuration options can be found on various sections of [Web developers documentation](#). For the full list, refer to [Web properties](#).

The most popular configuration options include:

- Debug mode, see `omero.web.debug`.
- Customizing OMERO clients e.g. to add your own logo to the login page (`omero.web.login_logo`) or use an index page as an alternative landing page for users (`omero.web.index_template`). See [OMERO.web UI customization](#) for further information.
- Enabling a public user see [Publishing data using OMERO.web](#).

Configuring Gunicorn

The following steps are run as the `omero-web` system user.

Additional settings can be configured by changing the properties below. Before changing the properties, run `export PATH=/opt/omero/web/venv3/bin:$PATH`:

- `omero.web.wsgi_workers` to $(2 \times \text{NUM_CORES}) + 1$

Note: Do not scale the number of workers to the number of clients you expect to have. OMERO.web should only need 4-12 worker processes to handle many requests per second.

- `omero.web.wsgi_args` Additional arguments. For more details check [Gunicorn Documentation](#). For example to enable **debugging**, run the following command:

```
omero config set omero.web.wsgi_args -- "--log-level=DEBUG --error-logfile=/opt/
↪omero/web/omero-web/var/log/error.log"
```

Setting up CORS

The following steps are run as root.

Cross Origin Resource Sharing allows web applications hosted at other origins to access resources from your OMERO.web installation. This can be achieved using the [django-cors-headers](#) app with additional configuration of OMERO.web. See the [django-cors-headers](#) page for more details on the settings.

Since OMERO.web 5.14.0, the package `django-cors-headers` is installed by default.

The following steps are run as the `omero-web` system user.

Configure CORS. An index is used to specify the ordering of middleware classes. It is important to add the `CorsMiddleware` as the first class and `CorsPostCsrfMiddleware` as the last. You can specify allowed origins in a whitelist, or allow all, for example:

```
omero config append omero.web.middleware '{"index": 0.5, "class": "corsheaders.
↪middleware.CorsMiddleware"}'
omero config append omero.web.middleware '{"index": 10, "class": "corsheaders.middleware.
↪CorsPostCsrfMiddleware"}'
omero config set omero.web.cors_origin_whitelist '["https://hostname.example.com"]'
# or to allow all
omero config set omero.web.cors_origin_allow_all True
```

Configuring NGINX

The following steps are run as root.

Copy the generated configuration file into the NGINX configuration directory, disable the default configuration and start NGINX:

```
sed -i.bak -re 's/( default_server.*)/; #\1/' /etc/nginx/nginx.conf
if [ -f /etc/nginx/conf.d/default.conf ]; then
    mv /etc/nginx/conf.d/default.conf /etc/nginx/conf.d/default.disabled
fi
cp /opt/omero/web/omero-web/nginx.conf.tmp /etc/nginx/conf.d/omeroweb.conf

systemctl enable nginx

systemctl start nginx
```

For production servers you may need to add additional directives to the configuration file, for example to enable [HTTPS](#). As an alternative to manually modifying the generated file you can generate a minimal configuration and include this in your own manually created NGINX file, such as `/etc/nginx/conf.d/omero-web.conf`:

```
omero web config nginx-location > /opt/omero/web/omero-web/omero-web-location.
↪include
```

This requires more initial work but in the future you can automatically regenerate your OMERO.web configuration and your additional configuration settings will still apply.

Note: If you need help configuring your firewall rules, see the [Server security and firewalls](#) page.

Running OMERO.web

Since OMERO.web 5.16.0, the package *whitenoise* is installed by default.

Optional: Install [Django Redis](#):

```
/opt/omero/web/venv3/bin/pip install 'django-redis==5.0.0'
```

The following steps are run as the omero-web system user.

Optional: Configure the cache:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.cache.RedisCache", "LOCATION": "redis://127.0.0.1:6379/0"}}'
omero config set omero.web.session_engine 'django.contrib.sessions.backends.cache'
```

Configure WhiteNoise and start OMERO.web manually to test the installation:

```
omero config append -- omero.web.middleware '{"index": 0, "class": "whitenoise.middleware.WhiteNoiseMiddleware"}'

omero web start

# Test installation e.g. curl -sL localhost:4080

omero web stop
```

Automatically running OMERO.web

The following steps are run as root.

Should you wish to run OMERO.web automatically, a *systemd.service* file could be created. See below an example file *omero-web-systemd.service*:

```
[Unit]
Description=OMERO.web
# Not mandatory, NGINX may be running on a different server
Requires=nginx.service
After=network.service

[Service]
User=omero-web
Type=forking
PIDFile=/opt/omero/web/omero-web/var/django.pid
Restart=no
RestartSec=10
Environment="PATH=/opt/omero/web/venv3/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin"
Environment="OMERODIR=/opt/omero/web/omero-web"
ExecStart=/opt/omero/web/venv3/bin/omero web start
ExecStop=/opt/omero/web/venv3/bin/omero web stop

[Install]
WantedBy=multi-user.target
```

Copy the *systemd.service* file, then enable and start the service:

```
cp omero-web-systemd.service /etc/systemd/system/omero-web.service

systemctl daemon-reload

systemctl enable omero-web.service

systemctl stop omero-web.service
```

(continues on next page)

```
systemctl start omero-web.service
```

Maintaining OMERO.web

The following steps are run as the omero-web system user.

You can manage the sessions using the following configuration options and commands:

- Session cookies `omero.web.session_expire_at_browser_close`:
 - A boolean that determines whether to expire the session when the user closes their browser. See [Django Browser-length sessions vs. persistent sessions documentation](#) for more details. The default value is `True`:

```
omero config set omero.web.session_expire_at_browser_close "True"
```

- The age of session cookies, in seconds. The default value is `86400`:

```
omero config set omero.web.session_cookie_age 86400
```

- Clear session:

Each session for a logged-in user in OMERO.web is kept in the session store. Stale sessions can cause the store to grow with time. OMERO.web uses by default the OS file system as the session store backend and does not automatically purge stale sessions, see [Django file-based session documentation](#) for more details. It is therefore the responsibility of the OMERO administrator to purge the session cache using the provided management command:

```
omero web clearsessions
```

It is recommended to call this command on a regular basis, for example as a daily cron job, see [Django clearing the session store documentation](#) for more information.

Troubleshooting

The following steps are run as the omero-web system user.

In order to identify why OMERO.web is not available run `omero web status`. Then consult `NGINX error.log` and `/opt/omero/web/omero-web/var/log/OMEROweb.log`.

Configuring Gunicorn advanced options

OMERO.web deployment can be configured with sync and async workers. **Sync workers** are faster and recommended for a data repository with [Download restrictions](#). If you wish to offer users the ability to download data then you have to use **async workers**. OMERO.web is able to handle multiple clients on a single worker thread switching context as necessary while streaming binary data from OMERO.server. Depending on the traffic and scale of the repository you should configure connections and speed limits on your server to avoid blocking resources. We recommend you run benchmark and performance tests. It is also possible to apply [Download restrictions](#) and offer alternative access to binary data.

Note: Handling streaming request/responses requires proxy buffering to be turned off. For more details refer to [Gunicorn deployment](#) and [NGINX configuration](#).

Note: `omero.web.application_server.max_requests` should be set to 0

See [Gunicorn design](#) for more details.

Experimental: Sync workers

The following steps are run as root.

Install <https://pypi.org/project/futures/>:

```
/opt/omero/web/venv3/bin/pip install futures
```

The following steps are run as the omero-web system user.

To find out more about the number of worker threads for handling requests, see [Gunicorn threads](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class
omero config set omero.web.wsgi_threads $(2-4 x NUM_CORES)
```

Experimental: Async workers

The following steps are run as root.

Install [Gevent](#) `>= 0.13`:

```
/opt/omero/web/venv3/bin/pip install 'gevent>=0.13'
```

The following steps are run as the omero-web system user.

To find out more about the maximum number of simultaneous clients, see [Gunicorn worker-connections](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class gevent
omero config set omero.web.wsgi_worker_connections 1000
omero config set omero.web.application_server.max_requests 0
```

SELinux

The following steps are run as root.

If you are running a system with [SELinux enabled](#) and are unable to access OMERO.web you may need to adjust the security policy:

```
if [ $(getenforce) != Disabled ]; then
    yum -y install policycoreutils-python
```

(continues on next page)

(continued from previous page)

```
setsebool -P httpd_read_user_content 1
setsebool -P httpd_enable_homedirs 1
semanage port -a -t http_port_t -p tcp 4080

fi
```

OMERO.web installation on Rocky Linux 9 and IcePy 3.6

Please first read *OMERO.server installation on RHEL 9/Rocky Linux 9*.

This is an example walkthrough for installing OMERO.web in a **virtual environment** using a dedicated system user. Installing OMERO.web in a virtual environment is the preferred way. For convenience in this walkthrough, we will use the **omero-web system user** and define the main OMERO.web configuration options as environment variables. Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process. By default, Python 3.9 is installed.

The following steps are run as root.

If required, first create a local system user omero-web and create directory:

```
useradd -m omero-web

mkdir -p /opt/omero/web/omero-web/etc/grid
chown -R omero-web /opt/omero/web/omero-web
```

Installing prerequisites

The following steps are run as root.

Install dependencies:

```
cp nginx.repo /etc/yum.repos.d/

dnf -y install unzip

dnf -y install python3

dnf -y install nginx
```

Optional: if you wish to use the Redis cache, install Redis:

```
dnf -y install redis

systemctl enable redis.service

systemctl start redis.service
```


Creating a virtual environment

The following steps are run as root.

Create the virtual environment. This is the recommended way to install OMERO.web:

```
python3 -mvenv /opt/omero/web/venv3
```

Install ZeroC IcePy 3.6:

```
/opt/omero/web/venv3/bin/pip install https://github.com/glencoesoftware/zeroc-ice-py-  
↪rhel9-x86_64/releases/download/20230830/zeroc_ice-3.6.5-cp39-cp39-linux_x86_64.whl
```

Upgrade pip and install OMERO.web:

```
/opt/omero/web/venv3/bin/pip install --upgrade pip  
/opt/omero/web/venv3/bin/pip install omero-web
```

Installing OMERO.web apps

A number of apps are available to add functionality to OMERO.web, such as [OMERO.figure](#) and [OMERO.iviewer](#). See the main website for a [list of released apps](#). These apps are optional and can be installed, as the **root user**, via **pip** to your OMERO.web virtual environment and configured as the **omero-web system user**, at any time.

Configuring OMERO.web

The following steps are run as the omero-web system user.

For convenience the main OMERO.web configuration options have been defined as environment variables. You can either use your own values, or alternatively use the following ones:

```
export WEBSSESSION=True  
export OMERODIR=/opt/omero/web/omero-web
```

Configure OMERO.web and create the NGINX OMERO configuration file to be included in a system-wide NGINX configuration by redirecting the output of the command `omero web config nginx` below into a file. If an attempt is made to access OMERO.web whilst it is not running, the generated NGINX configuration file will automatically display a maintenance page:

```
export PATH=/opt/omero/web/venv3/bin:$PATH  
  
omero web config nginx --http "${WEBSSESSION}" --servername "${WEBSSESSION_NAME}" > /opt/omero/  
↪web/omero-web/nginx.conf.tmp
```

OMERO.web offers a number of configuration options. The configuration changes **will not be applied** until Gunicorn is restarted using `omero web restart`. The Gunicorn workers are managed **separately** from other OMERO processes. You can check their status or stop them using `omero web status` or `omero web stop`.

- Session engine:
- OMERO.web offers alternative session backends to automatically delete stale data using the cache session store backend, see [Django cached session documentation](#) for more details.

- Redis requires `django-redis` in order to be used with OMERO.web. We assume that Redis has already been installed. To configure the cache, run:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.  
↪ cache.  
RedisCache", "LOCATION": "redis://127.0.0.1:6379/0"}}'
```

- After installing all the cache prerequisites set the following:

```
omero config set omero.web.session_engine django.contrib.sessions.backends.  
↪ cache
```

- Use a prefix:

By default OMERO.web expects to be run from the root URL of the webserver. This can be changed by setting `omero.web.prefix` and `omero.web.static_url`. For example, to make OMERO.web appear at `http://example.org/omero/`:

```
omero config set omero.web.prefix '/omero'  
omero config set omero.web.static_url '/omero/static/'
```

and regenerate your webserver configuration.

All configuration options can be found on various sections of [Web](#) developers documentation. For the full list, refer to [Web](#) properties.

The most popular configuration options include:

- Debug mode, see `omero.web.debug`.
- Customizing OMERO clients e.g. to add your own logo to the login page (`omero.web.login_logo`) or use an index page as an alternative landing page for users (`omero.web.index_template`). See [OMERO.web UI customization](#) for further information.
- Enabling a public user see [Publishing data using OMERO.web](#).

Configuring Gunicorn

The following steps are run as the `omero-web` system user.

Additional settings can be configured by changing the properties below. Before changing the properties, run `export PATH=/opt/omero/web/venv3/bin:$PATH`:

- `omero.web.wsgi_workers` to $(2 \times \text{NUM_CORES}) + 1$

Note: Do not scale the number of workers to the number of clients you expect to have. OMERO.web should only need 4-12 worker processes to handle many requests per second.

- `omero.web.wsgi_args` Additional arguments. For more details check [Gunicorn Documentation](#). For example to enable **debugging**, run the following command:

```
omero config set omero.web.wsgi_args -- "--log-level=DEBUG --error-logfile=/opt/  
↪ omero/web/omero-web/var/log/error.log"
```

Setting up CORS

The following steps are run as root.

Cross Origin Resource Sharing allows web applications hosted at other origins to access resources from your OMERO.web installation. This can be achieved using the [django-cors-headers](#) app with additional configuration of OMERO.web. See the [django-cors-headers](#) page for more details on the settings.

Since OMERO.web 5.14.0, the package *django-cors-headers* is installed by default.

The following steps are run as the omero-web system user.

Configure CORS. An index is used to specify the ordering of middleware classes. It is important to add the `CorsMiddleware` as the first class and `CorsPostCsrfMiddleware` as the last. You can specify allowed origins in a whitelist, or allow all, for example:

```
omero config append omero.web.middleware '{"index": 0.5, "class": "corsheaders.
↪middleware.CorsMiddleware"}'
omero config append omero.web.middleware '{"index": 10, "class": "corsheaders.middleware.
↪CorsPostCsrfMiddleware"}'
omero config set omero.web.cors_origin_whitelist '["https://hostname.example.com"]'
# or to allow all
omero config set omero.web.cors_origin_allow_all True
```

Configuring NGINX

The following steps are run as root.

Copy the generated configuration file into the NGINX configuration directory, disable the default configuration and start NGINX:

```
# Disable default server config
# Newer nginx versions:
if [ -f /etc/nginx/conf.d/default.conf ]; then
    mv /etc/nginx/conf.d/default.conf /etc/nginx/conf.d/default.disabled
fi
# Older nginx versions:
sed -i.bak -re 's/( default_server.*);#\1/' /etc/nginx/nginx.conf
sed -i "/^\s\s\s\sserver/, /\s\s\s\s\s/ s|^|#|" /etc/nginx/nginx.conf

cp /opt/omero/web/omero-web/nginx.conf.tmp /etc/nginx/conf.d/omeroweb.conf

systemctl enable nginx

systemctl start nginx
```

For production servers you may need to add additional directives to the configuration file, for example to enable [HTTPS](#). As an alternative to manually modifying the generated file you can generate a minimal configuration and include this in your own manually created NGINX file, such as `/etc/nginx/conf.d/omero-web.conf`:

```
omero web config nginx-location > /opt/omero/web/omero-web/omero-web-location.
↪include
```

This requires more initial work but in the future you can automatically regenerate your OMERO.web configuration and your additional configuration settings will still apply.

Note: If you need help configuring your firewall rules, see the *Server security and firewalls* page.

Running OMERO.web

Since OMERO.web 5.16.0, the package *whitenoise* is installed by default.

Optional: Install [Django Redis](#):

```
/opt/omero/web/venv3/bin/pip install 'django-redis==5.0.0'
```

The following steps are run as the omero-web system user.

Optional: Configure the cache:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.cache.RedisCache", "LOCATION": "redis://127.0.0.1:6379/0"}}'
omero config set omero.web.session_engine 'django.contrib.sessions.backends.cache'
```

Configure WhiteNoise and start OMERO.web manually to test the installation:

```
omero config append -- omero.web.middleware '{"index": 0, "class": "whitenoise.middleware.WhiteNoiseMiddleware"}'

omero web start

# Test installation e.g. curl -sL localhost:4080

omero web stop
```

Automatically running OMERO.web

The following steps are run as root.

Copy the *systemd.service* file, then enable and start the service:

```
cp omero-web-systemd.service /etc/systemd/system/omero-web.service

systemctl daemon-reload

systemctl enable omero-web.service

systemctl stop omero-web.service

systemctl start omero-web.service
```

Maintaining OMERO.web

The following steps are run as the omero-web system user.

You can manage the sessions using the following configuration options and commands:

- Session cookies `omero.web.session_expire_at_browser_close`:
 - A boolean that determines whether to expire the session when the user closes their browser. See [Django Browser-length sessions vs. persistent sessions documentation](#) for more details. The default value is `True`:

```
omero config set omero.web.session_expire_at_browser_close "True"
```

- The age of session cookies, in seconds. The default value is `86400`:

```
omero config set omero.web.session_cookie_age 86400
```

- Clear session:

Each session for a logged-in user in OMERO.web is kept in the session store. Stale sessions can cause the store to grow with time. OMERO.web uses by default the OS file system as the session store backend and does not automatically purge stale sessions, see [Django file-based session documentation](#) for more details. It is therefore the responsibility of the OMERO administrator to purge the session cache using the provided management command:

```
omero web clearsessions
```

It is recommended to call this command on a regular basis, for example as a daily cron job, see [Django clearing the session store documentation](#) for more information.

Troubleshooting

The following steps are run as the omero-web system user.

In order to identify why OMERO.web is not available run `omero web status`. Then consult `NGINX error.log` and `/opt/omero/web/omero-web/var/log/OMEROweb.log`.

Configuring Gunicorn advanced options

OMERO.web deployment can be configured with sync and async workers. **Sync workers** are faster and recommended for a data repository with [Download restrictions](#). If you wish to offer users the ability to download data then you have to use **async workers**. OMERO.web is able to handle multiple clients on a single worker thread switching context as necessary while streaming binary data from OMERO.server. Depending on the traffic and scale of the repository you should configure connections and speed limits on your server to avoid blocking resources. We recommend you run benchmark and performance tests. It is also possible to apply [Download restrictions](#) and offer alternative access to binary data.

Note: Handling streaming request/responses requires proxy buffering to be turned off. For more details refer to [Gunicorn deployment](#) and [NGINX configuration](#).

Note: `omero.web.application_server.max_requests` should be set to 0

See [Gunicorn design](#) for more details.

Experimental: Sync workers

The following steps are run as root.

Install <https://pypi.org/project/futures/>:

```
/opt/omero/web/venv3/bin/pip install futures
```

The following steps are run as the omero-web system user.

To find out more about the number of worker threads for handling requests, see [Gunicorn threads](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class  
omero config set omero.web.wsgi_threads $(2-4 x NUM_CORES)
```

Experimental: Async workers

The following steps are run as root.

Install [Gevent](#) `>= 0.13`:

```
/opt/omero/web/venv3/bin/pip install 'gevent>=0.13'
```

The following steps are run as the omero-web system user.

To find out more about the maximum number of simultaneous clients, see [Gunicorn worker-connections](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class gevent  
omero config set omero.web.wsgi_worker_connections 1000  
omero config set omero.web.application_server.max_requests 0
```

SELinux

The following steps are run as root.

If you are running a system with [SELinux enabled](#) and are unable to access OMERO.web you may need to adjust the security policy:

```
if [ $(getenforce) != Disabled ]; then  
    dnf -y install policycoreutils-python-utils  
    setsebool -P httpd_read_user_content 1  
    setsebool -P httpd_enable_homedirs 1  
    semanage port -a -t http_port_t -p tcp 4080  
    firewall-cmd --zone=public --add-port=4080/tcp --permanent  
    firewall-cmd --reload  
fi
```

OMERO.web installation on Ubuntu 20.04 and IcePy 3.6

Please first read *OMERO.server installation on Ubuntu 20.04*.

This is an example walkthrough for installing OMERO.web in a **virtual environment** using a dedicated system user. Installing OMERO.web in a virtual environment is the preferred way. For convenience in this walkthrough, we will use the **omero-web system user** and define the main OMERO.web configuration options as environment variables. Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process. By default, Python 3.8 is installed.

The following steps are run as root.

If required, first create a local system user omero-web and create directory:

```
useradd -m omero-web

mkdir -p /opt/omero/web/omero-web/etc/grid
chown -R omero-web /opt/omero/web/omero-web
```

Installing prerequisites

The following steps are run as root.

Install dependencies:

```
apt-get update

apt-get -y install unzip
apt-get -y install python3
apt-get -y install python3-venv

apt-get -y install nginx
```

Optional: if you wish to use the Redis cache, install Redis:

```
apt-get -y install redis-server

service redis-server start
```

Creating a virtual environment

The following steps are run as root.

Create the virtual environment. This is the recommended way to install OMERO.web:

```
python3 -mvenv /opt/omero/web/venv3
```

Install ZeroC IcePy 3.6:

```
/opt/omero/web/venv3/bin/pip install https://github.com/ome/zeroc-ice-ubuntu2004/
↳ releases/download/0.2.0/zeroc_ice-3.6.5-cp38-cp38-linux_x86_64.whl
```

Upgrade pip and install OMERO.web:

```
/opt/omero/web/venv3/bin/pip install --upgrade pip
/opt/omero/web/venv3/bin/pip install omero-web
```

Installing OMERO.web apps

A number of apps are available to add functionality to OMERO.web, such as [OMERO.figure](#) and [OMERO.iviewer](#). See the main website for a [list of released apps](#). These apps are optional and can be installed, as the **root user**, via **pip** to your OMERO.web virtual environment and configured as the **omero-web system user**, at any time.

Configuring OMERO.web

The following steps are run as the **omero-web system user**.

For convenience the main OMERO.web configuration options have been defined as environment variables. You can either use your own values, or alternatively use the following ones:

```
export WEBSSESSION=True
export OMERODIR=/opt/omero/web/omero-web
```

Configure OMERO.web and create the NGINX OMERO configuration file to be included in a system-wide NGINX configuration by redirecting the output of the command `omero web config nginx` below into a file. If an attempt is made to access OMERO.web whilst it is not running, the generated NGINX configuration file will automatically display a maintenance page:

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

omero web config nginx --http "${WEBPORT}" --servername "${WEBSERVER_NAME}" > /opt/omero/
↪web/omero-web/nginx.conf.tmp
```

OMERO.web offers a number of configuration options. The configuration changes **will not be applied** until Gunicorn is restarted using `omero web restart`. The Gunicorn workers are managed **separately** from other OMERO processes. You can check their status or stop them using `omero web status` or `omero web stop`.

- Session engine:
- OMERO.web offers alternative session backends to automatically delete stale data using the cache session store backend, see [Django cached session documentation](#) for more details.
- [Redis](#) requires [django-redis](#) in order to be used with OMERO.web. We assume that Redis has already been installed. To configure the cache, run:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.
↪cache.
RedisCache", "LOCATION": "redis://127.0.0.1:6379/0"}}'
```

- After installing all the cache prerequisites set the following:

```
omero config set omero.web.session_engine django.contrib.sessions.backends.
↪cache
```

- Use a prefix:

By default OMERO.web expects to be run from the root URL of the webserver. This can be changed by setting `omero.web.prefix` and `omero.web.static_url`. For example, to make OMERO.web appear at `http://example.org/omero/`:

```
omero config set omero.web.prefix '/omero'
omero config set omero.web.static_url '/omero/static/'
```

and regenerate your webserver configuration.

All configuration options can be found on various sections of [Web](#) developers documentation. For the full list, refer to [Web](#) properties.

The most popular configuration options include:

- Debug mode, see `omero.web.debug`.
- Customizing OMERO clients e.g. to add your own logo to the login page (`omero.web.login_logo`) or use an index page as an alternative landing page for users (`omero.web.index_template`). See [OMERO.web UI customization](#) for further information.
- Enabling a public user see [Publishing data using OMERO.web](#).

Configuring Gunicorn

The following steps are run as the `omero-web` system user.

Additional settings can be configured by changing the properties below. Before changing the properties, run `export PATH=/opt/omero/web/venv3/bin:$PATH`:

- `omero.web.wsgi_workers` to $(2 \times \text{NUM_CORES}) + 1$

Note: Do not scale the number of workers to the number of clients you expect to have. OMERO.web should only need 4-12 worker processes to handle many requests per second.

- `omero.web.wsgi_args` Additional arguments. For more details check [Gunicorn Documentation](#). For example to enable **debugging**, run the following command:

```
omero config set omero.web.wsgi_args -- "--log-level=DEBUG --error-logfile=/opt/
↪omero/web/omero-web/var/log/error.log"
```

Setting up CORS

The following steps are run as root.

Cross Origin Resource Sharing allows web applications hosted at other origins to access resources from your OMERO.web installation. This can be achieved using the [django-cors-headers](#) app with additional configuration of OMERO.web. See the [django-cors-headers](#) page for more details on the settings.

Since OMERO.web 5.14.0, the package `django-cors-headers` is installed by default.

The following steps are run as the `omero-web` system user.

Configure CORS. An index is used to specify the ordering of middleware classes. It is important to add the `CorsMiddleware` as the first class and `CorsPostCsrfMiddleware` as the last. You can specify allowed origins in a whitelist, or allow all, for example:

```
omero config append omero.web.middleware '{"index": 0.5, "class": "corsheaders.
↪middleware.CorsMiddleware"}'
omero config append omero.web.middleware '{"index": 10, "class": "corsheaders.middleware.
↪CorsPostCsrfMiddleware"}'
omero config set omero.web.cors_origin_whitelist '["https://hostname.example.com"]'
# or to allow all
omero config set omero.web.cors_origin_allow_all True
```

Configuring NGINX

The following steps are run as root.

Copy the generated configuration file into the NGINX configuration directory, disable the default configuration and start NGINX:

```
sed -i.bak -re 's/( default_server.*); #\1/' /etc/nginx/nginx.conf
rm /etc/nginx/sites-enabled/default
cp /opt/omero/web/omero-web/nginx.conf.tmp /etc/nginx/conf.d/omeroweb.conf

service nginx start
```

For production servers you may need to add additional directives to the configuration file, for example to enable [HTTPS](#). As an alternative to manually modifying the generated file you can generate a minimal configuration and include this in your own manually created NGINX file, such as `/etc/nginx/conf.d/omero-web.conf`:

```
omero web config nginx-location > /opt/omero/web/omero-web/omero-web-location.
↪include
```

This requires more initial work but in the future you can automatically regenerate your OMERO.web configuration and your additional configuration settings will still apply.

Note: If you need help configuring your firewall rules, see the [Server security and firewalls](#) page.

Running OMERO.web

Since OMERO.web 5.16.0, the package *whitenoise* is installed by default.

Optional: Install [Django Redis](#):

```
/opt/omero/web/venv3/bin/pip install 'django-redis==5.0.0'
```

The following steps are run as the omero-web system user.

Optional: Configure the cache:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.cache.RedisCache
↪", "LOCATION": "redis://127.0.0.1:6379/0"}}'
omero config set omero.web.session_engine 'django.contrib.sessions.backends.cache'
```

Configure WhiteNoise and start OMERO.web manually to test the installation:

```

omero config append -- omero.web.middleware '{"index": 0, "class": "whitenoise.
↳middleware.WhiteNoiseMiddleware"}'

omero web start

# Test installation e.g. curl -sL localhost:4080

omero web stop

```

Automatically running OMERO.web

The following steps are run as root.

Should you wish to run OMERO.web automatically, a *init.d* file could be created. See below an example file *omero-web-init.d*:

```

#!/bin/bash
#
# /etc/init.d/omero-web
# Subsystem file for "omero" web
#
### BEGIN INIT INFO
# Provides:          omero-web
# Required-Start:    $local_fs $remote_fs $network $time omero postgresql
# Required-Stop:     $local_fs $remote_fs $network $time omero postgresql
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: OMERO.web
### END INIT INFO
#
### Redhat
# chkconfig: - 98 02
# description: init file for OMERO.web
###

RETVAL=0
prog=omero-web

# Read configuration variable file if it is present
[ -r /etc/default/$prog ] && . /etc/default/$prog

OMERO_USER=${OMERO_USER:-omero-web}
OMERO=/opt/omero/web/venv3/bin/omero
OMERODIR=/opt/omero/web/omero-web
VENVDIR=${VENVDIR:-/opt/omero/web/venv3}

start() {
    echo -n "Starting $prog:"
    su - ${OMERO_USER} -c ". ${VENVDIR}/bin/activate;OMERODIR=${OMERODIR} ${OMERO} web_
↳start" &> /dev/null && echo -n ' OMERO.web'
    sleep 5
    RETVAL=$?
}

```

(continues on next page)

(continued from previous page)

```

    [ "$RETVAL" = 0 ]
        echo
}

stop() {
    echo -n "Stopping $prog:"
    su - ${OMERO_USER} -c ". ${VENVDIR}/bin/activate;OMERODIR=${OMERODIR} ${OMERO} web_
↪stop" &> /dev/null && echo -n ' Omero.web'
    RETVAL=$?
    [ "$RETVAL" = 0 ]
        echo
}

status() {
    echo -n "Status $prog:"
    su - ${OMERO_USER} -c ". ${VENVDIR}/bin/activate;OMERODIR=${OMERODIR} ${OMERO} web_
↪status"
    RETVAL=$?
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        stop
        start
        ;;
    status)
        status
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart|status}"
        RETVAL=1
esac
exit $RETVAL

```

Copy the *init.d* file, then configure the service:

```

cp omero-web-init.d /etc/init.d/omero-web
chmod a+x /etc/init.d/omero-web

update-rc.d -f omero-web remove
update-rc.d -f omero-web defaults 98 02

```

Start up services:

```
service redis-server start
```

(continues on next page)

(continued from previous page)

```
cron
service nginx start
service omero-web restart
```

Maintaining OMERO.web

The following steps are run as the omero-web system user.

You can manage the sessions using the following configuration options and commands:

- Session cookies `omero.web.session_expire_at_browser_close`:
 - A boolean that determines whether to expire the session when the user closes their browser. See [Django Browser-length sessions vs. persistent sessions documentation](#) for more details. The default value is `True`:

```
omero config set omero.web.session_expire_at_browser_close "True"
```

- The age of session cookies, in seconds. The default value is `86400`:

```
omero config set omero.web.session_cookie_age 86400
```

- Clear session:

Each session for a logged-in user in OMERO.web is kept in the session store. Stale sessions can cause the store to grow with time. OMERO.web uses by default the OS file system as the session store backend and does not automatically purge stale sessions, see [Django file-based session documentation](#) for more details. It is therefore the responsibility of the OMERO administrator to purge the session cache using the provided management command:

```
omero web clearsessions
```

It is recommended to call this command on a regular basis, for example as a daily cron job, see [Django clearing the session store documentation](#) for more information.

Troubleshooting

The following steps are run as the omero-web system user.

In order to identify why OMERO.web is not available run `omero web status`. Then consult `NGINX error.log` and `/opt/omero/web/omero-web/var/log/OMEROweb.log`.

Configuring Gunicorn advanced options

OMERO.web deployment can be configured with sync and async workers. **Sync workers** are faster and recommended for a data repository with [Download restrictions](#). If you wish to offer users the ability to download data then you have to use **async workers**. OMERO.web is able to handle multiple clients on a single worker thread switching context as necessary while streaming binary data from OMERO.server. Depending on the traffic and scale of the repository you should configure connections and speed limits on your server to avoid blocking resources. We recommend you run benchmark and performance tests. It is also possible to apply [Download restrictions](#) and offer alternative access to binary data.

Note: Handling streaming request/responses requires proxy buffering to be turned off. For more details refer to [Gunicorn deployment](#) and [NGINX configuration](#).

Note: `omero.web.application_server.max_requests` should be set to 0

See [Gunicorn design](#) for more details.

Experimental: Sync workers

The following steps are run as root.

Install <https://pypi.org/project/futures/>:

```
/opt/omero/web/venv3/bin/pip install futures
```

The following steps are run as the omero-web system user.

To find out more about the number of worker threads for handling requests, see [Gunicorn threads](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class  
omero config set omero.web.wsgi_threads $(2-4 x NUM_CORES)
```

Experimental: Async workers

The following steps are run as root.

Install [Gevent](#) `>= 0.13`:

```
/opt/omero/web/venv3/bin/pip install 'gevent>=0.13'
```

The following steps are run as the omero-web system user.

To find out more about the maximum number of simultaneous clients, see [Gunicorn worker-connections](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class gevent  
omero config set omero.web.wsgi_worker_connections 1000  
omero config set omero.web.application_server.max_requests 0
```

OMERO.web installation on Ubuntu 22.04 and IcePy 3.6

Please first read [OMERO.server installation on Ubuntu 22.04](#).

This is an example walkthrough for installing OMERO.web in a **virtual environment** using a dedicated system user. Installing OMERO.web in a virtual environment is the preferred way. For convenience in this walkthrough, we will use the **omero-web system user** and define the main OMERO.web configuration options as environment variables. Since 5.6, a new `OMERODIR` variable is used, you should first unset `OMERO_HOME` (if set) before beginning the installation process. By default, Python 3.10 is installed.

The following steps are run as root.

If required, first create a local system user omero-web and create directory:

```
useradd -m omero-web

mkdir -p /opt/omero/web/omero-web/etc/grid
chown -R omero-web /opt/omero/web/omero-web
```

Installing prerequisites

The following steps are run as root.

Install dependencies:

```
apt-get update

apt-get -y install unzip
apt-get -y install python3
apt-get -y install python3-venv

apt-get -y install nginx
```

Optional: if you wish to use the Redis cache, install Redis:

```
apt-get -y install redis-server

service redis-server start
```

Creating a virtual environment

The following steps are run as root.

Create the virtual environment. This is the recommended way to install OMERO.web:

```
python3 -mvenv /opt/omero/web/venv3
```

Install ZeroC IcePy 3.6:

```
/opt/omero/web/venv3/bin/pip install https://github.com/glencoesoftware/zeroc-ice-py-
↳ ubuntu2204-x86_64/releases/download/20221004/zeroc_ice-3.6.5-cp310-cp310-linux_x86_64.
↳ whl
```

Upgrade pip and install OMERO.web:

```
/opt/omero/web/venv3/bin/pip install --upgrade pip
/opt/omero/web/venv3/bin/pip install omero-web
```

Installing OMERO.web apps

A number of apps are available to add functionality to OMERO.web, such as [OMERO.figure](#) and [OMERO.iviewer](#). See the main website for a [list of released apps](#). These apps are optional and can be installed, as the **root user**, via **pip** to your OMERO.web virtual environment and configured as the **omero-web system user**, at any time.

Configuring OMERO.web

The following steps are run as the **omero-web system user**.

For convenience the main OMERO.web configuration options have been defined as environment variables. You can either use your own values, or alternatively use the following ones:

```
export WEBSSESSION=True
export OMERODIR=/opt/omero/web/omero-web
```

Configure OMERO.web and create the NGINX OMERO configuration file to be included in a system-wide NGINX configuration by redirecting the output of the command `omero web config nginx` below into a file. If an attempt is made to access OMERO.web whilst it is not running, the generated NGINX configuration file will automatically display a maintenance page:

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

omero web config nginx --http "${WEBPORT}" --servername "${WEBSERVER_NAME}" > /opt/omero/
↪web/omero-web/nginx.conf.tmp
```

OMERO.web offers a number of configuration options. The configuration changes **will not be applied** until Gunicorn is restarted using `omero web restart`. The Gunicorn workers are managed **separately** from other OMERO processes. You can check their status or stop them using `omero web status` or `omero web stop`.

- Session engine:
- OMERO.web offers alternative session backends to automatically delete stale data using the cache session store backend, see [Django cached session documentation](#) for more details.
- [Redis](#) requires [django-redis](#) in order to be used with OMERO.web. We assume that Redis has already been installed. To configure the cache, run:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.
↪cache.
RedisCache", "LOCATION": "redis://127.0.0.1:6379/0"}}'
```

- After installing all the cache prerequisites set the following:

```
omero config set omero.web.session_engine django.contrib.sessions.backends.
↪cache
```

- Use a prefix:

By default OMERO.web expects to be run from the root URL of the webserver. This can be changed by setting `omero.web.prefix` and `omero.web.static_url`. For example, to make OMERO.web appear at `http://example.org/omero/`:

```
omero config set omero.web.prefix '/omero'
omero config set omero.web.static_url '/omero/static/'
```


and regenerate your webserver configuration.

All configuration options can be found on various sections of [Web](#) developers documentation. For the full list, refer to [Web](#) properties.

The most popular configuration options include:

- Debug mode, see `omero.web.debug`.
- Customizing OMERO clients e.g. to add your own logo to the login page (`omero.web.login_logo`) or use an index page as an alternative landing page for users (`omero.web.index_template`). See [OMERO.web UI customization](#) for further information.
- Enabling a public user see [Publishing data using OMERO.web](#).

Configuring Gunicorn

The following steps are run as the `omero-web` system user.

Additional settings can be configured by changing the properties below. Before changing the properties, run `export PATH=/opt/omero/web/venv3/bin:$PATH`:

- `omero.web.wsgi_workers` to $(2 \times \text{NUM_CORES}) + 1$

Note: Do not scale the number of workers to the number of clients you expect to have. OMERO.web should only need 4-12 worker processes to handle many requests per second.

- `omero.web.wsgi_args` Additional arguments. For more details check [Gunicorn Documentation](#). For example to enable **debugging**, run the following command:

```
omero config set omero.web.wsgi_args -- "--log-level=DEBUG --error-logfile=/opt/
↳omero/web/omero-web/var/log/error.log"
```

Setting up CORS

The following steps are run as root.

Cross Origin Resource Sharing allows web applications hosted at other origins to access resources from your OMERO.web installation. This can be achieved using the [django-cors-headers](#) app with additional configuration of OMERO.web. See the [django-cors-headers](#) page for more details on the settings.

Since OMERO.web 5.14.0, the package *django-cors-headers* is installed by default.

The following steps are run as the `omero-web` system user.

Configure CORS. An index is used to specify the ordering of middleware classes. It is important to add the `CorsMiddleware` as the first class and `CorsPostCsrfMiddleware` as the last. You can specify allowed origins in a whitelist, or allow all, for example:

```
omero config append omero.web.middleware '{"index": 0.5, "class": "corsheaders.
↳middleware.CorsMiddleware"}'
omero config append omero.web.middleware '{"index": 10, "class": "corsheaders.middleware.
↳CorsPostCsrfMiddleware"}'
omero config set omero.web.cors_origin_whitelist '["https://hostname.example.com"]'
# or to allow all
omero config set omero.web.cors_origin_allow_all True
```

Configuring NGINX

The following steps are run as root.

Copy the generated configuration file into the NGINX configuration directory, disable the default configuration and start NGINX:

```
sed -i.bak -re 's/( default_server.*); #\1/' /etc/nginx/nginx.conf
rm /etc/nginx/sites-enabled/default
cp /opt/omero/web/omero-web/nginx.conf.tmp /etc/nginx/conf.d/omeroweb.conf

service nginx start
```

For production servers you may need to add additional directives to the configuration file, for example to enable [HTTPS](#). As an alternative to manually modifying the generated file you can generate a minimal configuration and include this in your own manually created NGINX file, such as `/etc/nginx/conf.d/omero-web.conf`:

```
omero web config nginx-location > /opt/omero/web/omero-web/omero-web-location.
↪include
```

This requires more initial work but in the future you can automatically regenerate your OMERO.web configuration and your additional configuration settings will still apply.

Note: If you need help configuring your firewall rules, see the [Server security and firewalls](#) page.

Running OMERO.web

Since OMERO.web 5.16.0, the package *whitenoise* is installed by default.

Optional: Install [Django Redis](#):

```
/opt/omero/web/venv3/bin/pip install 'django-redis==5.0.0'
```

The following steps are run as the omero-web system user.

Optional: Configure the cache:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.cache.RedisCache"
↪, "LOCATION": "redis://127.0.0.1:6379/0"}}'
omero config set omero.web.session_engine 'django.contrib.sessions.backends.cache'
```

Configure WhiteNoise and start OMERO.web manually to test the installation:

```
omero config append -- omero.web.middleware '{"index": 0, "class": "whitenoise.
↪middleware.WhiteNoiseMiddleware"}'

omero web start

# Test installation e.g. curl -sL localhost:4080

omero web stop
```

Automatically running OMERO.web

The following steps are run as root.

Should you wish to run OMERO.web automatically, a *init.d* file could be created. See below an example file *omero-web-init.d*:

```
#!/bin/bash
#
# /etc/init.d/omero-web
# Subsystem file for "omero" web
#
### BEGIN INIT INFO
# Provides:          omero-web
# Required-Start:    $local_fs $remote_fs $network $time omero
# Required-Stop:    $local_fs $remote_fs $network $time omero
# Default-Start:    2 3 4 5
# Default-Stop:     0 1 6
# Short-Description: OMERO.web
### END INIT INFO
#
### Redhat
# chkconfig: - 98 02
# description: init file for OMERO.web
###

RETVAL=0
prog=omero-web

# Read configuration variable file if it is present
[ -r /etc/default/$prog ] && . /etc/default/$prog

OMERO_USER=${OMERO_USER:-omero-web}
OMERO=/opt/omero/web/venv3/bin/omero
OMERODIR=/opt/omero/web/omero-web
VENVDIR=${VENVDIR:-/opt/omero/web/venv3}

start() {
    echo -n "Starting $prog:"
    su - ${OMERO_USER} -c ". ${VENVDIR}/bin/activate;OMERODIR=${OMERODIR} ${OMERO} web_
↪start" &> /dev/null && echo -n ' OMERO.web'
    sleep 5
    RETVAL=$?
    [ "$RETVAL" = 0 ]
    echo
}

stop() {
    echo -n "Stopping $prog:"
    su - ${OMERO_USER} -c ". ${VENVDIR}/bin/activate;OMERODIR=${OMERODIR} ${OMERO} web_
↪stop" &> /dev/null && echo -n ' OMERO.web'
    RETVAL=$?
    [ "$RETVAL" = 0 ]
    echo
}
```

(continues on next page)

(continued from previous page)

```

}

status() {
    echo -n $"Status $prog:"
    su - ${OMERO_USER} -c ". ${VENVDIR}/bin/activate;OMERODIR=${OMERODIR} ${OMERO} web_
↪status"
    RETVAL=$?
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        stop
        start
        ;;
    status)
        status
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart|status}"
        RETVAL=1
esac
exit $RETVAL

```

Copy the *init.d* file, then configure the service:

```

cp omero-web-init.d /etc/init.d/omero-web
chmod a+x /etc/init.d/omero-web

update-rc.d -f omero-web remove
update-rc.d -f omero-web defaults 98 02

```

Start up services:

```

service redis-server start

cron
service nginx start
service omero-web restart

```

Maintaining OMERO.web

The following steps are run as the `omero-web` system user.

You can manage the sessions using the following configuration options and commands:

- Session cookies `omero.web.session_expire_at_browser_close`:
 - A boolean that determines whether to expire the session when the user closes their browser. See [Django Browser-length sessions vs. persistent sessions documentation](#) for more details. The default value is `True`:

```
omero config set omero.web.session_expire_at_browser_close "True"
```

- The age of session cookies, in seconds. The default value is `86400`:

```
omero config set omero.web.session_cookie_age 86400
```

- Clear session:

Each session for a logged-in user in OMERO.web is kept in the session store. Stale sessions can cause the store to grow with time. OMERO.web uses by default the OS file system as the session store backend and does not automatically purge stale sessions, see [Django file-based session documentation](#) for more details. It is therefore the responsibility of the OMERO administrator to purge the session cache using the provided management command:

```
omero web clearsessions
```

It is recommended to call this command on a regular basis, for example as a daily cron job, see [Django clearing the session store documentation](#) for more information.

Troubleshooting

The following steps are run as the `omero-web` system user.

In order to identify why OMERO.web is not available run `omero web status`. Then consult `NGINX error.log` and `/opt/omero/web/omero-web/var/log/OMEROweb.log`.

Configuring Gunicorn advanced options

OMERO.web deployment can be configured with sync and async workers. **Sync workers** are faster and recommended for a data repository with [Download restrictions](#). If you wish to offer users the ability to download data then you have to use **async workers**. OMERO.web is able to handle multiple clients on a single worker thread switching context as necessary while streaming binary data from OMERO.server. Depending on the traffic and scale of the repository you should configure connections and speed limits on your server to avoid blocking resources. We recommend you run benchmark and performance tests. It is also possible to apply [Download restrictions](#) and offer alternative access to binary data.

Note: Handling streaming request/responses requires proxy buffering to be turned off. For more details refer to [Gunicorn deployment](#) and [NGINX configuration](#).

Note: `omero.web.application_server.max_requests` should be set to 0

See [Gunicorn design](#) for more details.

Experimental: Sync workers

The following steps are run as root.

Install <https://pypi.org/project/futures/>:

```
/opt/omero/web/venv3/bin/pip install futures
```

The following steps are run as the omero-web system user.

To find out more about the number of worker threads for handling requests, see [Gunicorn threads](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class  
omero config set omero.web.wsgi_threads $(2-4 x NUM_CORES)
```

Experimental: Async workers

The following steps are run as root.

Install [Gevent](#) ≥ 0.13 :

```
/opt/omero/web/venv3/bin/pip install 'gevent>=0.13'
```

The following steps are run as the omero-web system user.

To find out more about the maximum number of simultaneous clients, see [Gunicorn worker-connections](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class gevent  
omero config set omero.web.wsgi_worker_connections 1000  
omero config set omero.web.application_server.max_requests 0
```

OMERO.web installation on Debian 10 and IcePy 3.6

Please first read *[OMERO.server installation on Debian 10](#)*.

This is an example walkthrough for installing OMERO.web in a **virtual environment** using a dedicated system user. Installing OMERO.web in a virtual environment is the preferred way. For convenience in this walkthrough, we will use the **omero-web system user** and define the main OMERO.web configuration options as environment variables. Since 5.6, a new OMERODIR variable is used, you should first unset OMERO_HOME (if set) before beginning the installation process. By default, Python 3.6 is installed.

The following steps are run as root.

If required, first create a local system user omero-web and create directory:

```
useradd -m omero-web  
  
mkdir -p /opt/omero/web/omero-web/etc/grid  
chown -R omero-web /opt/omero/web/omero-web
```

Installing prerequisites

The following steps are run as root.

Install dependencies:

```
apt-get update

apt-get -y install unzip

apt-get -y install python3
apt-get -y install python3-venv

apt-get -y install nginx
```

Optional: if you wish to use the Redis cache, install Redis:

```
apt-get -y install redis-server

service redis-server start
```

Creating a virtual environment

The following steps are run as root.

Create the virtual environment. This is the recommended way to install OMERO.web:

```
python3 -mvenv /opt/omero/web/venv3
```

Install ZeroC IcePy 3.6:

```
/opt/omero/web/venv3/bin/pip install https://github.com/ome/zeroc-ice-debian10/releases/
↳download/0.1.0/zeroc_ice-3.6.5-cp37-cp37m-linux_x86_64.whl
```

Upgrade pip and install OMERO.web:

```
/opt/omero/web/venv3/bin/pip install --upgrade pip
/opt/omero/web/venv3/bin/pip install omero-web
```

Installing OMERO.web apps

A number of apps are available to add functionality to OMERO.web, such as [OMERO.figure](#) and [OMERO.iviewer](#). See the main website for a [list of released apps](#). These apps are optional and can be installed, as the **root user**, via **pip** to your OMERO.web virtual environment and configured as the **omero-web system user**, at any time.

Configuring OMERO.web

The following steps are run as the `omero-web` system user.

For convenience the main OMERO.web configuration options have been defined as environment variables. You can either use your own values, or alternatively use the following ones:

```
export WEBSESSION=True
export OMERODIR=/opt/omero/web/omero-web
```

Configure OMERO.web and create the NGINX OMERO configuration file to be included in a system-wide NGINX configuration by redirecting the output of the command `omero web config nginx` below into a file. If an attempt is made to access OMERO.web whilst it is not running, the generated NGINX configuration file will automatically display a maintenance page:

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

omero web config nginx --http "${WEBPORT}" --servername "${WEBSERVER_NAME}" > /opt/omero/
↪web/omero-web/nginx.conf.tmp
```

OMERO.web offers a number of configuration options. The configuration changes **will not be applied** until Gunicorn is restarted using `omero web restart`. The Gunicorn workers are managed **separately** from other OMERO processes. You can check their status or stop them using `omero web status` or `omero web stop`.

- Session engine:
- OMERO.web offers alternative session backends to automatically delete stale data using the cache session store backend, see [Django cached session documentation](#) for more details.
- Redis requires [django-redis](#) in order to be used with OMERO.web. We assume that Redis has already been installed. To configure the cache, run:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.
↪cache.
RedisCache", "LOCATION": "redis://127.0.0.1:6379/0"}}'
```

- After installing all the cache prerequisites set the following:

```
omero config set omero.web.session_engine django.contrib.sessions.backends.
↪cache
```

- Use a prefix:

By default OMERO.web expects to be run from the root URL of the webserver. This can be changed by setting `omero.web.prefix` and `omero.web.static_url`. For example, to make OMERO.web appear at `http://example.org/omero/`:

```
omero config set omero.web.prefix '/omero'
omero config set omero.web.static_url '/omero/static/'
```

and regenerate your webserver configuration.

All configuration options can be found on various sections of [Web developers documentation](#). For the full list, refer to [Web properties](#).

The most popular configuration options include:

- Debug mode, see `omero.web.debug`.

- Customizing OMERO clients e.g. to add your own logo to the login page (`omero.web.login_logo`) or use an index page as an alternative landing page for users (`omero.web.index_template`). See [OMERO.web UI customization](#) for further information.
- Enabling a public user see [Publishing data using OMERO.web](#).

Configuring Gunicorn

The following steps are run as the `omero-web` system user.

Additional settings can be configured by changing the properties below. Before changing the properties, run `export PATH=/opt/omero/web/venv3/bin:$PATH`:

- `omero.web.wsgi_workers` to $(2 \times \text{NUM_CORES}) + 1$

Note: Do not scale the number of workers to the number of clients you expect to have. OMERO.web should only need 4-12 worker processes to handle many requests per second.

- `omero.web.wsgi_args` Additional arguments. For more details check [Gunicorn Documentation](#). For example to enable **debugging**, run the following command:

```
omero config set omero.web.wsgi_args -- "--log-level=DEBUG --error-logfile=/opt/
↳omero/web/omero-web/var/log/error.log"
```

Setting up CORS

The following steps are run as root.

Cross Origin Resource Sharing allows web applications hosted at other origins to access resources from your OMERO.web installation. This can be achieved using the [django-cors-headers](#) app with additional configuration of OMERO.web. See the [django-cors-headers](#) page for more details on the settings.

Since OMERO.web 5.14.0, the package *django-cors-headers* is installed by default.

The following steps are run as the `omero-web` system user.

Configure CORS. An index is used to specify the ordering of middleware classes. It is important to add the `CorsMiddleware` as the first class and `CorsPostCsrfMiddleware` as the last. You can specify allowed origins in a whitelist, or allow all, for example:

```
omero config append omero.web.middleware '{"index": 0.5, "class": "corsheaders.
↳middleware.CorsMiddleware"}'
omero config append omero.web.middleware '{"index": 10, "class": "corsheaders.middleware.
↳CorsPostCsrfMiddleware"}'
omero config set omero.web.cors_origin_whitelist '["https://hostname.example.com"]'
# or to allow all
omero config set omero.web.cors_origin_allow_all True
```

Configuring NGINX

The following steps are run as root.

Copy the generated configuration file into the NGINX configuration directory, disable the default configuration and start NGINX:

```
sed -i.bak -re 's/( default_server.*); #\1/' /etc/nginx/nginx.conf
rm /etc/nginx/sites-enabled/default
cp /opt/omero/web/omero-web/nginx.conf.tmp /etc/nginx/conf.d/omeroweb.conf

service nginx start
```

For production servers you may need to add additional directives to the configuration file, for example to enable [HTTPS](#). As an alternative to manually modifying the generated file you can generate a minimal configuration and include this in your own manually created NGINX file, such as `/etc/nginx/conf.d/omero-web.conf`:

```
omero web config nginx-location > /opt/omero/web/omero-web/omero-web-location.
↪include
```

This requires more initial work but in the future you can automatically regenerate your OMERO.web configuration and your additional configuration settings will still apply.

Note: If you need help configuring your firewall rules, see the [Server security and firewalls](#) page.

Running OMERO.web

Since OMERO.web 5.16.0, the package *whitenoise* is installed by default.

Optional: Install [Django Redis](#):

```
/opt/omero/web/venv3/bin/pip install 'django-redis==5.0.0'
```

The following steps are run as the omero-web system user.

Optional: Configure the cache:

```
omero config set omero.web.caches '{"default": {"BACKEND": "django_redis.cache.RedisCache",
↪"LOCATION": "redis://127.0.0.1:6379/0"}}'
omero config set omero.web.session_engine 'django.contrib.sessions.backends.cache'
```

Configure WhiteNoise and start OMERO.web manually to test the installation:

```
omero config append -- omero.web.middleware '{"index": 0, "class": "whitenoise.
↪middleware.WhiteNoiseMiddleware"}'

omero web start

# Test installation e.g. curl -sL localhost:4080

omero web stop
```

Automatically running OMERO.web

The following steps are run as root.

Should you wish to run OMERO.web automatically, a *init.d* file could be created. See below an example file *omero-web-init.d*:

```
#!/bin/bash
#
# /etc/init.d/omero-web
# Subsystem file for "omero" web
#
### BEGIN INIT INFO
# Provides:          omero-web
# Required-Start:     $local_fs $remote_fs $network $time omero postgresql
# Required-Stop:      $local_fs $remote_fs $network $time omero postgresql
# Default-Start:      2 3 4 5
# Default-Stop:       0 1 6
# Short-Description:  OMERO.web
### END INIT INFO
#
# chkconfig: - 98 02
# description: init file for OMERO.web
###

RETVAL=0
prog=omero-web

# Read configuration variable file if it is present
[ -r /etc/default/$prog ] && . /etc/default/$prog

OMERO_USER=${OMERO_USER:-omero-web}
OMERO=/opt/omero/web/venv3/bin/omero
OMERODIR=/opt/omero/web/omero-web
VENVDIR=${VENVDIR:-/opt/omero/web/venv3}

start() {
    echo -n "Starting $prog:"
    su - ${OMERO_USER} -c ". ${VENVDIR}/bin/activate;OMERODIR=${OMERODIR} ${OMERO} web_
↪start" &> /dev/null && echo -n ' OMERO.web'
    sleep 5
    RETVAL=$?
    [ "$RETVAL" = 0 ]
    echo
}

stop() {
    echo -n "Stopping $prog:"
    su - ${OMERO_USER} -c ". ${VENVDIR}/bin/activate;OMERODIR=${OMERODIR} ${OMERO} web_
↪stop" &> /dev/null && echo -n ' OMERO.web'
    RETVAL=$?
    [ "$RETVAL" = 0 ]
    echo
}
}
```

(continues on next page)

(continued from previous page)

```
status() {
    echo -n "Status $prog:"
    su - ${OMERO_USER} -c ". ${VENVDIR}/bin/activate;OMERODIR=${OMERODIR} ${OMERO} web_
↪status"
    RETVAL=$?
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        stop
        start
        ;;
    status)
        status
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart|status}"
        RETVAL=1
esac
exit $RETVAL
```

Copy the *init.d* file, then configure the service:

```
cp omero-web-init.d /etc/init.d/omero-web
chmod a+x /etc/init.d/omero-web

update-rc.d -f omero-web remove
update-rc.d -f omero-web defaults 98 02
```

Start up services:

```
service redis-server start

service nginx start
service omero-web restart
```

Maintaining OMERO.web

The following steps are run as the `omero-web` system user.

You can manage the sessions using the following configuration options and commands:

- Session cookies `omero.web.session_expire_at_browser_close`:
 - A boolean that determines whether to expire the session when the user closes their browser. See [Django Browser-length sessions vs. persistent sessions documentation](#) for more details. The default value is `True`:

```
omero config set omero.web.session_expire_at_browser_close "True"
```

- The age of session cookies, in seconds. The default value is `86400`:

```
omero config set omero.web.session_cookie_age 86400
```

- Clear session:

Each session for a logged-in user in OMERO.web is kept in the session store. Stale sessions can cause the store to grow with time. OMERO.web uses by default the OS file system as the session store backend and does not automatically purge stale sessions, see [Django file-based session documentation](#) for more details. It is therefore the responsibility of the OMERO administrator to purge the session cache using the provided management command:

```
omero web clearsessions
```

It is recommended to call this command on a regular basis, for example as a daily cron job, see [Django clearing the session store documentation](#) for more information.

Troubleshooting

The following steps are run as the `omero-web` system user.

In order to identify why OMERO.web is not available run `omero web status`. Then consult `NGINX error.log` and `/opt/omero/web/omero-web/var/log/OMEROweb.log`.

Configuring Gunicorn advanced options

OMERO.web deployment can be configured with sync and async workers. **Sync workers** are faster and recommended for a data repository with [Download restrictions](#). If you wish to offer users the ability to download data then you have to use **async workers**. OMERO.web is able to handle multiple clients on a single worker thread switching context as necessary while streaming binary data from OMERO.server. Depending on the traffic and scale of the repository you should configure connections and speed limits on your server to avoid blocking resources. We recommend you run benchmark and performance tests. It is also possible to apply [Download restrictions](#) and offer alternative access to binary data.

Note: Handling streaming request/responses requires proxy buffering to be turned off. For more details refer to [Gunicorn deployment](#) and [NGINX configuration](#).

Note: `omero.web.application_server.max_requests` should be set to 0

See [Gunicorn design](#) for more details.

Experimental: Sync workers

The following steps are run as root.

Install <https://pypi.org/project/futures/>:

```
/opt/omero/web/venv3/bin/pip install futures
```

The following steps are run as the omero-web system user.

To find out more about the number of worker threads for handling requests, see [Gunicorn threads](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class  
omero config set omero.web.wsgi_threads $(2-4 x NUM_CORES)
```

Experimental: Async workers

The following steps are run as root.

Install [Gevent](#) >= 0.13:

```
/opt/omero/web/venv3/bin/pip install 'gevent>=0.13'
```

The following steps are run as the omero-web system user.

To find out more about the maximum number of simultaneous clients, see [Gunicorn worker-connections](#). Additional settings can be configured by changing the following properties:

```
omero config set omero.web.wsgi_worker_class gevent  
omero config set omero.web.wsgi_worker_connections 1000  
omero config set omero.web.application_server.max_requests 0
```

Note: Support for Apache deployment has been dropped in 5.3.0.

If your organization's policies only allow Apache to be used as the external-facing web-server you should configure Apache to proxy connections to an NGINX instance running on your OMERO server i.e. use Apache as a reverse proxy. For more details see [Apache mod_proxy documentation](#).

2.2.3 OMERO.server binary repository

About

The OMERO.server binary data repository is a fundamental piece of server-side functionality. It provides optimized and indexed storage of original file, pixel and thumbnail data, attachments and full-text indexes. The repository's directories contain various files that, together with your SQL database, constitute the information about your users and their data that OMERO.server relies upon for normal operation.

Layout

The repository is internally laid out as follows:

```
/OMERO
/OMERO/Pixels          <--- Pixel data and pyramids
/OMERO/Files           <--- Original file data
/OMERO/Thumbnails      <--- Thumbnail data
/OMERO/FullText        <--- Lucene full text search index
/OMERO/ManagedRepository <--- OMERO.fs filesets, with import logs
/OMERO/BioFormatsCache <--- Cached Bio-Formats state for rendering
```

Your repository is not:

- the “database”
- the directory where your OMERO.server binaries are
- the directory where your OMERO.client (OMERO.insight or OMERO.importer) binaries are
- your PostgreSQL data directory

PixelService resolution order for locating binary data for images

When the server is trying to find the binary data for an image, it looks:

- first under */OMERO/Pixels* for a *\$NUMBER_pyramid* file
- then under */OMERO/Pixels* for a regular *\$NUMBER* file
- then under */OMERO/Files* for OMERO 4 files
- or under */OMERO/ManagedRepository* for OMERO 5 files

Locking and remote shares

The OMERO server requires proper locking semantics on all files in the binary repository. In practice, this means that remotely mounted shares such as AFS, CIFS, and NFS can cause issues. If you have experience and/or the time to manage and monitor the locking implementations of your remote filesystem, then using them as for your binary repository should be fine.

If, however, you are seeing errors such as `NullPointerException`, “Bad file descriptors” and similar in your server log, then you will need to use directly connected disks.

Warning: If your binary repository is a remote share and mounting the share fails or is dismounted, OMERO will continue operating using the mount point instead! To prevent this, make the mount point read-only for the OMERO user so that no data can be written to the mount point.

Changing your repository location

Note: It is **strongly** recommended that you make all changes to your OMERO binary repository with the server shut down. Changing the `omero.data.dir` configuration does **not** move the repository for you, you must do this yourself.

Your repository location can be changed from its `/OMERO` default by modifying your `OMERO.server` configuration as follows:

```
$ omero config set omero.data.dir /mnt/really_big_disk/OMERO
```

The suggested procedure is to shut down your `OMERO.server` instance, move your repository, change your `omero.data.dir` and then start the instance back up. For example:

```
$ omero admin stop
$ mv /OMERO /mnt/really_big_disk
$ omero config set omero.data.dir /mnt/really_big_disk/OMERO
$ omero admin start
```

The `omero.managed.dir` property for the `OMERO.fs` managed repository may be adjusted similarly, even to a directory outside `omero.data.dir`.

Note: The managed repository should be located and configured to allow the OMERO server processes fast access to the uploaded filesets that it contains.

Access permissions

Your repository should be owned by the same user that is starting your `OMERO.server` instance. This is often either yourself (find this out by executing `whoami`) or a separate `omero` (or similar) user who is dedicated to running `OMERO.server`. For example:

```
$ whoami
omero
$ ls -al /OMERO
total 24
drwxr-xr-x  5 omero omero 128 Dec 12 2006 .
drwxr-xr-x  7 root  root 160 Nov  5 15:24 ..
drwxr-xr-x  3 omero omero 4096 Dec 20 10:13 BioFormatsCache
drwxr-xr-x  2 omero omero 1656 Dec 18 14:31 Files
drwxr-xr-x 150 omero omero 12288 Dec 20 10:00 ManagedRepository
drwxr-xr-x 25 omero omero 23256 Dec 10 19:06 Pixels
drwxr-xr-x  2 omero omero  48 Dec  8 2006 Thumbnails
```


Repository size

At minimum, the binary repository should be comfortably larger than the images and other files that users may be uploading to it. It is fine to set `omero.data.dir` or `omero.managed.dir` to very large volumes, or to use logical volume management to conveniently increase space as necessary.

2.2.4 OMERO.server and PostgreSQL

In order to be installed, OMERO.server requires a running PostgreSQL instance that is configured to accept connections over TCP. This section explains how to ensure that you have the correct PostgreSQL version and that it is installed and configured correctly.

Ensuring you have a valid PostgreSQL version

For OMERO 5.6, PostgreSQL version 14 or later is recommended. Make sure you are using a [supported version](#).

You can check which version of PostgreSQL you have installed with any of the following commands:

```
$ createuser -V
createuser (PostgreSQL) 9.4.1
$ psql -V
psql (PostgreSQL) 9.4.1
$ createdb -V
createdb (PostgreSQL) 9.4.1
```

If your existing PostgreSQL installation is an earlier version, it is recommended that you upgrade to a more up-to-date version. Before upgrading, stop the OMERO server and then perform a full dump of the database using **pg_dump**. See the [OMERO.server backup and restore](#) section for further details.

If using a Linux distribution-provided PostgreSQL server, upgrading to a newer version of the distribution will usually make a newer version of PostgreSQL available. If the database was not migrated to the new version automatically, restore your backup after installing, configuring and starting the new version of the database server. If a PostgreSQL server was not provided by your system, [EnterpriseDB](#) provide an installer.

Checking PostgreSQL port listening status

You can check if PostgreSQL is listening on the default port (TCP/5432) by running the following command:

```
$ netstat -an | egrep '5432.*LISTEN'
tcp        0      0 0.0.0.0:5432          0.0.0.0:*           LISTEN
tcp        0      0 :::5432              :::*                 LISTEN
```

Note: The exact output of this command will vary. The important thing to recognize is whether or not a process is listening on TCP/5432.

If you cannot find a process listening on TCP/5432 you will need to find your `postgresql.conf` file and enable PostgreSQL's TCP listening mode. The exact location of the `postgresql.conf` file varies between installations.

It may be helpful to locate it using the package manager (`rpm` or `dpkg`) or by utilizing the `find` command. Usually, the PostgreSQL data directory (which houses the `postgresql.conf` file, is located under `/var` or `/usr`:

```
$ sudo find /etc -name 'postgresql.conf'
$ sudo find /usr -name 'postgresql.conf'
$ sudo find /var -name 'postgresql.conf'
/var/lib/postgresql/data/postgresql.conf
```

Note: The PostgreSQL data directory is usually only readable by the user `postgres` so you will likely have to be root in order to find it.

Once you have found the location of the `postgresql.conf` file on your particular installation, you will need to enable TCP listening. The area of the configuration file you are concerned about should look similar to this:

```
#listen_addresses = 'localhost'          # what IP address(es) to listen on;
                                           # comma-separated list of addresses;
                                           # defaults to 'localhost', '*' = all

#port = 5432
max_connections = 100
# note: increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction). You
# might also need to raise shared_buffers to support more connections.
#superuser_reserved_connections = 2
#unix_socket_directory = *
#unix_socket_group = *
#unix_socket_permissions = 0777          # octal
#bonjour_name = *                        # defaults to the computer name
```

PostgreSQL HBA (host based authentication)

OMERO.server must have permission to connect to the database that has been created in your PostgreSQL instance. This is configured in the *host based authentication* file, `pg_hba.conf`. Check the configuration by examining the contents of `pg_hba.conf`. It's important that at least one line allows connections from the loopback address (`127.0.0.1`) as follows:

#	TYPE	DATABASE	USER	CIDR-ADDRESS	METHOD
# IPv4 local connections:					
host	all	all	127.0.0.1/32	md5	
# IPv6 local connections:					
host	all	all	::1/128	md5	

Note: The other lines that are in your `pg_hba.conf` are important either for PostgreSQL internal commands to work or for existing applications you may have. **Do not delete them.**

Completing configuration

After making any configuration changes to `postgresql.conf` or `pg_hba.conf`, reload the server for the changes to take effect.

```
$ sudo service postgresql reload
```

See also:

PostgreSQL

Interactive documentation for the current release of PostgreSQL.

Connections and Authentication

Section of the PostgreSQL documentation about configuring the server using *postgresql.conf*.

Client Authentication

Chapter of the PostgreSQL documentation about configuring client authentication with *pg_hba.conf*.

2.2.5 Installing additional features

OMERO.grid

To unify the various components of OMERO, OMERO.grid was developed to monitor and control processes over numerous remote systems. Based on [ZeroC's](#) IceGrid framework, OMERO.grid provides management access, distributed background processing, log handling and several other features.

Terminology

Please notice that ZeroC uses a specific naming scheme for IceGrid elements and actors. A *server* in the context of this document is not a host computer - it is a process running inside an IceGrid node, servicing incoming requests. A *host* is a computer on which IceGrid elements get deployed. For more details, see [Terminology](#).

Getting started

Requirements

The normal OMERO installation actually makes use of OMERO.grid internally. If you have followed the instructions under [OMERO.server installation](#) you will have everything you need to start working with OMERO.grid.

The standard install should also be used to install other hosts in the grid, such as a computation-only host. Some elements can be omitted for a computation-only host such as PostgreSQL, Apache/nginx, etc.

Running OMERO.web and/or starting up the full OMERO.server instance is not required in such a case (only the basic requirements to run **omero node** are needed, i.e. ZeroC Ice and Python modules for OMERO scripts).

IceGrid Tools

If you would like to explore your IceGrid configuration, use

```
omero admin ice
```

It provides full access to the **icegridadmin** console described in the ZeroC manual. Specific commands can also be executed:

```
omero admin ice help
omero admin ice application list
omero admin ice application describe OMERO
omero admin ice server list
```

Further, by running **java -jar ice-gridgui.jar** the GUI provided by ZeroC can be used to administer OMERO.grid. This JAR is provided in the OMERO source code under `lib/repository`.

See also:

icegridadmin Command Line Tool

Chapter of the [ZeroC](#) manual about the **icegridadmin** CLI

IceGrid GUI Tool

Chapter of the [ZeroC](#) manual about the IceGrid GUI tool

How it works

IceGrid is a location and activation service, which functions as a central registry to manage all your OMERO server processes. OMERO.grid provides server components which use the registry to communicate with one another. Other than a minimal amount of configuration and starting a single daemon on each host machine, OMERO.grid manages the complexity of all your computing resources.

Deployment descriptors

All the resources for a single OMERO site are described by one **application descriptor**. OMERO ships with several example descriptors under `etc/grid`. These descriptors describe what processes will be started on what nodes, identified by simple names. For example the [default descriptor](#), used if no other file is specified, defines the **master** node. As you will see, these files are critical both for the correct functioning of your server as well as its security.

The deployment descriptors provided define which server instances are started on which nodes. The default descriptor configures the **master** node to start the *OMERO.blitz* server, the Glacier2 router for firewalling, as well as a single processor - **Processor0**. The master node is also configured via `etc/master.cfg` to host the registry, though this process can be started elsewhere.

Deployment commands

The **master** node must be started first to provide the registry. This is done via the **omero admin start** command which uses the default descriptor:

```
omero admin start
```

The **deploy** command looks for any changes to the defined descriptor and restarts only those servers which have modifications:

```
omero admin deploy
```

Both **omero admin start** and **omero admin deploy** can optionally take a path to an application descriptor which must be passed on every invocation:

```
omero admin deploy etc/grid/my-site.xml
```

Two other nodes, then, each provide a single processor, **Processor1** and **Processor2**. These are started via:

To start a node identified by **NAME**, the following command can be used

```
omero node start NAME
```

At this point the node will try and connect to the registry to announce its presence. If a node with the same name is already started, then registration will fail, which is important to prevent unauthorized users.

The configuration of your grid, however, is very much up to you. Based on the example descriptor files (*.xml) and configuration files (*.cfg), it is possible to develop OMERO.grid installations completely tailored to your computing resources.

The whole grid can be shutdown by stopping the master node via: **omero admin stop**. Each individual node can also be shutdown via: **omero node NAME stop** on that particular node.

Deployment examples

Two examples will be presented showing the flexibility of OMERO.grid deployment and identifying files whose modification is critical for the deployment to work.

Nodes on a single host

The first example will focus on changing the deployed nodes/servers on a single host. It should serve as an introduction to the concepts. Unless used for very specific requirements, this type of deployment doesn't yield any performance gains.

The first change that you will want to make to your application descriptor is to add additional processors. Take a look at <https://github.com/ome/openmicroscopy/blob/develop/etc/templates/grid/default.xml>. There you can define two new nodes - **node1** and **node2** by simply adding a new XML element below the master node definition:

```
<node name="node1">
  <server-instance template="ProcessorTemplate" index="1"/>
</node>

<node name="node2">
```

(continues on next page)

(continued from previous page)

```
<server-instance template="ProcessorTemplate" index="2"/>
</node>
```

Remember to change the node name and the index number for each subsequent node definition. The node name and the index number do not need to match. In fact, the index number can be completely ignored, except for the fact that it must be unique. The node name, however, is important for properly starting your new processor.

You will need both a configuration file under `etc/` with the same name, and unless the node name matches the name of your local host, you will need to specify it on the command line:

```
omero node node1 start
```

or with the environment variable `OMERO_NODE`:

```
OMERO_NODE=node1 omero node start
```

After starting up both nodes, you can verify that you now have three processors running by looking at the output of **omero admin diagnostics**.

For more information on using scripts, see the *OMERO.scripts advanced topics*.

Nodes on multiple hosts

Warning: Before attempting this type of deployment, make sure that the hosts can ping each other and that required ports are open and not firewalled.

A more complex deployment example is running multiple nodes on networked hosts. Initially, the host's loopback IP address (127.0.0.1) is used in the grid configuration files.

For this example, let's presume we have control over two hosts: `omero-master` (IP address 192.168.0.1/24) and `omero-slave` (IP address 192.168.0.2/24). The goal is to move the processor server onto another host (`omero-slave`) to reduce the load on the host running the master node (`omero-master`). The configuration changes required to achieve this are outlined below.

On host `omero-master`:

- `etc/grid/default.xml` - remove or comment out from the master node the `server-instance` using the `ProcessorTemplate`. Below the master node add an XML element defining a new node:

```
<node name="omero-slave">
  <server-instance template="ProcessorTemplate" index="0" dir="" />
</node>
```

- `etc/internal.cfg` - change the value of `Ice.Default Locator` from 127.0.0.1 to 192.168.0.1
- `etc/master.cfg` - change all occurrences of 127.0.0.1 to 192.168.0.1

On host `omero-slave`:

- copy or rename `etc/node1.cfg` to `etc/omero-slave.cfg` and change all `node1` strings to `omero-slave` in `etc/omero-slave.cfg`. Also update the `IceGrid.Node.Endpoints` value to `tcp -h 192.168.0.2`
- `etc/internal.cfg` - change the value of `Ice.Default Locator` from 127.0.0.1 to 192.168.0.1
- `etc/ice.config` - add the line `Ice.Default.Router=OMERO.Glacier2/router:tcp -p 4063 -h 192.168.0.1`

To apply the changes, start the OMERO instance on the `omero-master` node by using **`omero admin start`**. After that, start the `omero-slave` node by using **`omero node omero-slave start`**. Issuing **`omero admin diagnostics`** on the master node should show a running processor instance and the `omero-slave` node should accept job requests from the master node.

Securing grid resources

More than just making sure no malicious code enters your grid, it is critical to prevent unauthorized access via the application descriptors (*.xml) and configuration (*.cfg) as mentioned above.

Firewall

The simplest and most effective way of preventing unauthorized access is to have all OMERO.grid resources behind a firewall. Only the Glacier2 router has a port visible to machines outside the firewall. If this is possible in your configuration, then you can leave the internal endpoints unsecured.

SSL (Secure Socket Layer)

Though it is probably unnecessary to use transport encryption within a firewall, encryption from clients to the Glacier2 router will often be necessary. For more information on SSL, see [SSL](#).

Permissions Verifier

The IceSSL plugin can be used both for encrypting the channel as well as authenticating users. SSL-based authentication, however, can be difficult to configure especially for within the firewall, and so instead you may want to configure a “permissions verifier” to prevent non-trusted users from accessing a system within your firewall. From `master.cfg`:

```
IceGrid.Registry.AdminPermissionsVerifier=IceGrid/NullPermissionsVerifier
#IceGrid.Registry.AdminCryptPasswords=etc/passwd
```

Here we have defined a “null” permissions verifier which allows anyone to connect to the registry’s administrative endpoints. One simple way of securing these endpoints is to use the `AdminCryptPasswords` property, which expects a passwd-formatted file at the given relative or absolute path:

```
mrmypasswordisomero TN7CjkTVoDnb2
msmypasswordisome jkyZ3t9JXPRRU
```

where these values come from using `openssl`:

```
$ openssl
OpenSSL> passwd
Password:
Verifying - Password:
TN7CjkTVoDnb2
OpenSSL>
```

Another possibility is to use the [OMERO.blitz](#) permissions verifier, so that anyone with a proper OMERO account can access the server.

See [Controlling Access to IceGrid Sessions](#) of the Ice manual for more information.

Unique node names

Only a limited number of node names are configured in an application descriptor. For an unauthorized user to fill a slot, they must know the name (which **is** discoverable with the right code) and be the first to contact the grid saying “I am Node029”, for example. A system administrator need only then be certain that all the node slots are taken up by trusted machines and users.

It is also possible to allow “dynamic registration” in which servers are added to the registry after the fact. In some situations this may be quite useful, but is disabled by default. Before enabling it, be sure to have secured your endpoints via one of the methods outlined above.

Absolute paths

The example application descriptors shipped with OMERO all use relative paths to make installation easier. Once you are comfortable with configuring OMERO.grid, it would most likely be safer to configure absolute paths. For example, specifying that nodes execute under `/usr/lib/omero` requires that whoever starts the node have access to that directory. Therefore, as long as you control the boxes which can attach to your endpoints (see [Firewall](#)), then you can be relatively certain that no tampering can occur with the installed binaries.

Technical information and other tips

Processes

It is important to understand just what processes will be running on your servers. When you run **omero admin start**, **icegridnode** is executed which starts a controlling daemon and deploys the proper descriptor. This configuration is persisted under `var/master` and `var/registry`.

Once the application is loaded, the **icegridnode** daemon process starts up all the servers which are configured in the descriptor. If one of the processes fails, it will be restarted. If restart fails, eventually the server will be “disabled”. On shutdown, the **icegridnode** process also shutdowns all the server processes.

Targets

In application descriptors, it is possible to surround sections of the description with `<target/>` elements. For example, in `templates.xml` the section which defines the main *OMERO.blitz* server includes:

```
<server id="Blitz-{index}" exe="{JAVA}" activation="always" pwd="{OMERO_HOME}">
  <target name="debug">
    <option>-Xdebug</option>
    <option>-Xrunjdp:server=y,transport=dt_socket,address=8787,suspend=y</option>
  </target>
  . . .
```

When the application is deployed, if “debug” is added as a target, then the `-Xdebug`, etc. options will be passed to the Java runtime. This will allow remote connection to your server over the configured port.

Multiple targets can be enabled at the same time:

```
omero admin deploy etc/grid/default.xml debug secure someothertarget
```


Ice.MessageSizeMax

Ice imposes an upper limit on all method invocations. This limit, `Ice.MessageSizeMax`, is configured in your application descriptor (e.g. `templates.xml`) and configuration files (e.g. `ice.config`). The setting must be applied to all servers which will be handling the invocation. For example, a call to `InteractiveProcessor.execute(omero::RMap inputs)` which passes the inputs all the way down to `processor.py` will need to have a sufficiently large `Ice.MessageSizeMax` for: the client, the Glacier2 router, the *OMERO.blitz* server, and the Processor.

The default is currently set to 65536 kilobytes which is 64MB.

Logging

Currently all output from OMERO.grid is stored in `$OMERO_PREFIX/var/log/master.out` with error messages going to `$OMERO_PREFIX/var/log/master.err`. Individual services may also create their own log files.

Shortcuts

If the `omero` script is copied or symlinked to another name, then the script will separate the name on hyphens and execute `omero` with the second and later parts **prepended** to the argument list.

For example,

```
ln -s omero omero-admin
omero-admin start
```

works identically to:

```
omero admin start
```

Symbolic linking

Shortcuts allow the `bin/omero` script to function as an `init.d` script when named `omero-admin`, and need only be copied to `/etc/init.d/` to function properly. It will resolve its installation directory, and execute from there.

For example,

```
ln -s $VENV_SERVER/bin/omero /usr/local/bin/omero
omero-admin start
```

The same works for putting `bin/omero` on your path:

```
PATH=$VENV_SERVER/bin:$PATH
```

This means that OMERO.grid can be unpacked anywhere, and as long as the user invoking the commands has the proper permissions on the `$OMERO_PREFIX` directory, it will function normally.

Running as root

One exception to this rule is that starting OMERO.grid as root may actually delegate to another user, if the “user” attribute is set on the <server/> elements in `etc/grid/templates.xml`.

See also:

OMERO sessions

OMERO.mail

The OMERO server has the ability to send email to any users who have a properly configured email address. OMERO system administrators can then use the **omero admin email** command to contact those users.

In order to activate the subsystem, minimally the `omero.mail.config` property will need to be activated. It is likely you will need to change the defaults for the following connection properties:

- `omero.mail.host`
- `omero.mail.port`
- `omero.mail.smtp.auth`
- `omero.mail.smtp.starttls.enable`
- `omero.mail.from`

All properties can be found under the *Mail* section of *Configuration properties glossary*.

Note: A current limitation of the system is that emails are not in a queue and therefore if you log out or otherwise lose your OMERO session before the server has finished sending, the action will abort without completing.

Example secure SMTP configurations

Replace `omero@gmail.com` and `mypassword` with your real credentials.

Send email via GMail using TLS (port 587):

```
omero.mail.config=true
omero.mail.from=omero@gmail.com
omero.mail.host=smtp.googlemail.com
omero.mail.port=587
omero.mail.smtp.auth=true
omero.mail.username=omero@gmail.com
omero.mail.password=mypassword
omero.mail.smtp.starttls.enable=true
```

Send email via GMail using SSL (port 465):

```
omero.mail.config=true
omero.mail.from=omero@gmail.com
omero.mail.host=smtp.googlemail.com
omero.mail.port=465
omero.mail.smtp.auth=true
omero.mail.username=omero@gmail.com
```

(continues on next page)

(continued from previous page)

```
omero.mail.password=mypassword
omero.mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
```

Example minimum configuration

```
$ omero config set omero.mail.config true
```

By default, this will use localhost as the mail server on port 25 and send as the user **omero**.

To use your actual mail server:

```
$ omero config set omero.mail.host smtp.university.example
```

If authentication is required, then also configure:

```
$ omero config set omero.mail.username USER
$ omero config set omero.mail.password PASS
```

Setting email addresses

For any user to receive email, a valid email address must be configured. By default, the *root* OMERO user will *not* have an email address configured. This can be done from one of the UIs or via the **omero obj** command:

```
$ omero obj update Experimenter:0 email=root@university.example
```

Note: Using a mailing list or an alias for the *root* user can simplify configuration.

Enabling mail notifications

A number of “mail senders” are available for sending notifications of certain events on the server. Those available include:

- **ServerUpMailSender** and **ServerDownMailSender** which mail when the server goes up or down
- **FailedLoginMailSender** which can be configured to send for particular users if a bad password is used
- **ObjectMailSender** which can be configured to send an email under various conditions. Instances which are configured include:
 - **newUserMailSender** which sends an email every time a user is created
 - **newCommentMailSender** which sends an email every time a user’s image is commented on by another user

To activate the senders, the `etc/blitz/mail-senders.example` can be copied to a file ending with “.xml”.

OMERO.web error reporting

OMERO.web will email the users listed in the `omero.web.admins` whenever the application identify broken link (HTTP status code 404) or raises an unhandled exception that results in an internal server error (HTTP status code 500). This gives the administrators immediate notification of any errors. The `omero.web.admins` will get a description of the error, a complete Python traceback, and details about the HTTP request that caused the error.

Note: Reporting errors requires property `omero.web.debug` set to `False` and works together with [OMERO.web error handling](#).

Further configuration

Finally, if the above mail configuration properties do not cover your needs, you can add your own implementation as described under [Extending OMERO.server](#). The related property is `omero.mail.bean`:

```
$ omero config set omero.mail.bean myMailImplementation
```

OMERO.movie

A short decription on how to create movies from OMERO.

Creating a movie from OMERO

OMERO provides a script to make Mpeg or Quicktime movies from any image in the server. These movies are created by a script called `makemovie.py`, this script has a number of options: these include: selecting a range of Z,T planes, the channels to display. The movie can also show information overlayed over the image: z-section, scale bar and timing.

The resulting movie will then be uploaded to the server by the script and become a file attachment to the source image.

Viewing the movie

The make movie script allows you to save the movie in two different formats, a DivX-encoded AVI and QuickTime movie. To view the AVI you may need to install a DivX codec from [DivX](#). It should be noted that the DivX AVI is normally 1/3 to 1/10 the size of the QuickTime movie.

Installing the make movie script

The make movie script currently uses the [mencoder](#) utility to encode the movies, this command should be in the path of the computer (icgrid node) running the script.

We have [Mac OSX installs for mencoder](#) which were originally provided [here](#). Unzip and put the mencoder in the PATH available to the server, e.g. `/usr/local/bin/`. You may need to restart the server for this to take effect.

There are also macports, rpms and deps for mencoder.

Make movie also uses [Pillow](#) and [numpy](#).

Make movie command arguments

A detailed list of the commands accepted by the script are:

- `imageId`: This id of the image to create the movie from
- `output`: The name of the output file, sans the extension
- `zStart`: The starting z-section to create the movie from
- `zEnd`: The final z-section
- `tStart`: The starting timepoint to create the movie
- `tEnd`: The final timepoint.
- `channels`: The list of channels to use in the movie (index, from 0)
- `splitView`: Should we show the split view in the movie (not available yet)
- `showTime`: Show the average time of the acquisition of the channels in the frame.
- `showPlaneInfo`: Show the time and z-section of the current frame.
- `fps`: The number of frames per second of the movie
- `scalebar`: The scalebar size in microns, if ≤ 0 will not show scale bar.
- `format`: The format of the movie to be created currently supports 'video/mpeg', 'video/quicktime'
- `overlayColour`: The colour of the overlays, scalebar, time, as int(RGB)
- `fileAnnotation`: The fileAnnotation id of the uploaded movie. (return value from script)

OMERO.scripts

OMERO.scripts are the OME version of plugins, allowing you to extend the functionality of OMERO. Official core OMERO.scripts come bundled with every OMERO.server release but you can also add new scripts you have written yourself or found via the repositories forked from [ome/omero-user-scripts](#).

Prerequisites

Uploading and managing scripts

OMERO.scripts user guide describes the workflow for developing and uploading scripts as an Admin. **Any scripts you add to the `lib/scripts/` directory as a server admin will be considered 'trusted' and automatically detected by OMERO, allowing them to be run on the server from the clients or command line by any of your users.**

Once in the directory, scripts cannot be automatically updated and any additional ones will be lost when you upgrade your server installation. Therefore, we recommend you use a Github repository to manage your scripts. If you are not familiar with [using git](#), you can use the [GitHub app for your OS](#) (available for Mac and Windows but not Linux). The basic workflow is:

- fork our [omero-user-script](#) repository
- clone it in your `lib/scripts` directory

```
cd lib/scripts;
git clone git@github.com:YOURGITUSERNAME/omero-user-scripts.git YOUR_SCRIPTS
```

- save the scripts you want to use into the appropriate sub-directory in your cloned location `lib/scripts/YOUR_SCRIPTS`

Then when you upgrade your OMERO.server installation, provided your Github repository is up to date with all your latest script versions (i.e. all your local changes are committed), you just need to repeat the `git clone` step. Those scripts will then be automatically detected by your new server installation and available for use from the clients and command line as before.

Client Server SSL verification

If you configure OMERO.web behind NGINX with a recognized SSL certificate your users can be sure that they are connecting to their intended server.

OMERO.server and clients do not automatically support host verification, so a [man-in-the-middle attack](#) is possible. This may result in users inadvertently transmitting their login credentials to an attacker.

This can be remedied by configuring OMERO.server with a certificate and ensuring all OMERO clients are configured to verify the server certificate before connecting.

Server certificate

The easiest solution is to use the [omero-certificates](#) plugin to generate self-signed server certificates alongside their associated configuration. This workflow is described in the particular sections of [OMERO.server installation](#) documentation.

Here we describe an alternative option to the usage of the [omero-certificates](#) plugin. This option is re-using the SSL certificates used to protect OMERO.web. First convert the public certificate `server.pem` and private key `server.key` to the PKCS12 format where `secret` is the password used to protect the combined output file `server.p12`:

```
openssl pkcs12 -export -out server.p12 -in server.pem -inkey server.key -passout ↵  
↵pass:secret
```

Copy `server.p12` to the OMERO.server host, for instance to `/etc/ssl/omero/`.

External access to OMERO.server is managed by the Glacier2 component which can be configured as follows:

```
omero config set omero.glacier2.IceSSL.Ciphers HIGH  
# Look for certificates in this directory, you can omit and use the full path to files ↵  
↵instead  
omero config set omero.glacier2.IceSSL.DefaultDir /etc/ssl/omero/  
omero config set omero.glacier2.IceSSL.CertFile server.p12  
omero config set omero.glacier2.IceSSL.Password secret  
omero config set omero.glacier2.IceSSL.Protocols tls1_2  
omero config set omero.glacier2.IceSSL.ProtocolVersionMin tls1_2  
omero config set omero.glacier2.IceSSL.ProtocolVersionMax tls1_2
```

Restart OMERO.server.

Internal certificate authority

You can also create your own certificates by creating a certificate authority (CA), and using that to create a server certificate. Set this additional server configuration property to point to the public CA certificate `/etc/ssl/omero/cacert.pem`:

```
omero config set omero.glacier2.IceSSL.CAs cacert.pem
```

ZeroC provide the [Ice Certificate Utilities package](#) to help create certificates, but if you know what you are doing you can use `openssl` directly.

Client host verification

At present there is no easy way to configure the standard OMERO clients to require host verification.

If you are a developer the following Ice properties can be passed to the `omero.client` constructor to force host validation:

- `IceSSL.Ciphers=HIGH`
- `IceSSL.VerifyPeer=1`
- `IceSSL.VerifyDepthMax=0`
- `IceSSL.UsePlatformCAs=1`
- `IceSSL.Protocols=tls1_2` (if required by the server configuration)

Some platforms or languages do not support the cipher specification `HIGH`. Instead you can specify a cipher family such as `AES256` or `AES_256`. See the [IceSSL.Ciphers documentation](#).

If you have your own certificate authority replace `IceSSL.UsePlatformCAs` with:

- `IceSSL.CAs=/path/to/CA/cacert.pem`

These properties check that the certificate chain is valid, but they do not verify that the hostname matches that of the certificate. To verify the hostname either set:

- `IceSSL.CheckCertName=1`

If your certificate hostname does not match exactly (for example, if you have a wildcard certificate) use the `IceSSL.TrustOnly` property instead. Multiple CN can be specified:

- `IceSSL.TrustOnly=CN=omero.example.org;CN=*.example.org`

Further information

- <https://doc.zeroc.com/technical-articles/glacier2-articles/teach-yourself-glacier2-in-10-minutes#TeachYourselfGlacier2in10Minutes-UsingSSLwithGlacier2>
- <https://doc.zeroc.com/ice/3.6/ice-plugins/icessl/configuring-icessl>
- <https://doc.zeroc.com/ice/3.6/ice-plugins/icessl/setting-up-a-certificate-authority>
- <https://doc.zeroc.com/ice/3.6/property-reference/icessl>

OMERO.server Websockets

OMERO 5.5.0 includes experimental support for websocket connections. This allows clients to connect to OMERO.server over HTTP/S using the Ice protocol (note: this is not the same as the *OMERO.web* or *JSON APIs*).

Configuration

The `omero.client.icetransports` OMERO.server configuration property must be changed. See the linked documentation for details.

You can override the default `ws` (4065) and `wss` (4066) ports with the properties `omero.ports.ws` `omero.ports.wss`.

If you want to proxy OMERO.server websockets via a webserver such as Nginx you must also add a cipher supported by Nginx to `omero.glacier2.IceSSL.Ciphers` since the anonymous ciphers that OMERO uses are not supported.

For a full configuration example see <https://github.com/ome/docker-example-omero-websockets>

Client connection

You can connect to an OMERO websocket by setting the appropriate `Ice.Config` properties in the *client*, for example:

```
Ice.Default.Router="OMERO.Glacier2/router:wss -p 8443 -h example.org -r omero/websocket"
```

Some clients also support specifying the Ice transport in the host, e.g. `wss://example.org:8443/omero/websocket`.

2.3 Upgrading

Starting with OMERO 5.6, OMERO.server and OMERO.web installations are assumed to be separate throughout documentation, each with its own virtualenv. and installation directory.

2.3.1 Migrate to a new operating system

This is an example walkthrough for migrating an OMERO.server to a new operating system.

Basic steps

1. Choose a platform. If your current installation platform does not match one of the *recommended platforms*, you may want to choose a new platform as your migration target. See *Choosing a platform* below.
2. If you choose to Install OMERO.server and OMERO.web *separately* (optionally, you can install both on a single device): It will be preferable to choose the same platform type for both installations.
3. Once both have been installed, perform an OMERO.server *Backup and restore* procedure.

Choosing a platform

The two recommended platforms are RHEL/RockyLinux 9 and Ubuntu 22.04.

Installation

The installation walkthroughs provided in the documentation covers a minimum installation only. For this, check the main *server* and *web* installation pages.

However, more advanced installation mechanisms are available if you are interested and have familiarity with the given mechanism:

- **Ansible roles** are available for most installation steps. The primary roles, *omero-server* and *omero-web* allow to install respectively the OMERO.server and OMERO.web.
- **Docker images** are also available. We offer production quality images of both *omero-server* and *omero-web*.

Please get in touch at <https://forum.image.sc/tag/omero> if you have any questions.

Backup and restore

Minimally, you will need to back up and restore the binary repository and the database.

Backup

- Perform the backups following the *backup instructions*.
- When ready, copy them to the new operating system:

```
rsync -ra --progress db.dump NEW_OPERATING_SYSTEM:/tmp
rsync -ra --progress OMERO_BINARY_BACKUP NEW_OPERATING_SYSTEM:/tmp
```

On the new operating system, stop the OMERO.server.

Restore the binary

- Remove all the directories under /OMERO e.g. `sudo rm -rf /OMERO/*`.
- Move the backup diretory containing the `omero.config`, `/tmp/OMERO_BINARY_BACKUP/backup`, out of `/tmp/OMERO_BINARY_BACKUP`
- Change ownership and group of the directories under `OMERO_BINARY_BACKUP`:

```
sudo chown -R omero-server /tmp/OMERO_BINARY_BACKUP/
```

- As the `omero-server` user, move all the backup directories under `/tmp/OMERO_BINARY_BACKUP` to `/OMERO`, for example:

```
sudo -u omero-server -s
mv /tmp/OMERO_BINARY_BACKUP/ManagedRepository /OMERO/ManagedRepository
```

Restore the database

- Restore the database backup following the *restore PostgreSQL instructions*.

Restore the OMERO.server configuration

- Restore the backup configuration:

```
omero config load /tmp/OMERO_BINARY_BACKUP/backup/omero.config
```

You should then follow the *Configuration* steps of *OMERO.server installation* to adjust your configuration as/if necessary.

Virtual environments update

When installing the OMERO.server on the new operating system, a virtual environment `/opt/omero/server/venv3` is created. If you have installed some specific packages in the virtual environment on the **previous** operating system, the packages will need to be re-installed in virtual environment on the **new** operating system.

The same is true for the OMERO.web virtual environment.

Scripts

If you have added or modified scripts under `lib/scripts` on the **previous** operating system, you will need to back up your modifications and re-implement the modifications under `lib/scripts` on the **new** operating system, See *Merge script changes*

Certificates

See *Server certificate*

Restart your server

- Following a successful migration upgrade, you can start the server.

```
$ omero admin start
```

- If anything goes wrong, please send the output of `omero admin diagnostics` to the [forum](#).

Reference *OMERO.server upgrade*, *OMERO.server backup and restore*.

2.3.2 Migration to Python 3

Basic steps

1. Choose a platform and a Python version. If your current installation platform does not match one of the *recommended platforms*, you may want to choose a new platform as your migration target. See *Choosing a platform* below.

2. Install OMERO.server and OMERO.web *separately*. Though not necessary, all instructions like *OMERO.server* and *OMERO.web* below as well as the main *server* and *web* installation pages now assume that the two are in separate installations.
3. Once both have been installed, perform a *backup and restore* procedure and test your installation against the copy of your data.

Choosing a platform

The two recommended platforms, CentOS 7 and Ubuntu 18.04, have Python 3.6 as default installation and have therefore received the most testing which is why Python 3.6 is the preferred version of Python.

Both Python 3.5 and 3.7 should work and are slated to have support added, but Python 3.6 has been the focus of testing during the migration.

Similarly, other operating systems are slated for having support added, but help from the community would be very welcome! Obvious next candidates are CentOS 8 and Ubuntu 20.04.

Debian 9 is still on Python 3.5 and Debian 10 has moved to Ice 3.7. We have nonetheless an installation guide for Debian 9 with Python 3.5 and Ice 3.6 but an installation guide for Debian 10 with Python 3.7 and Ice 3.6.

Other prerequisites

OMERO's other prerequisites have not changed substantially but if you would like to take this opportunity to move to the *recommended version* for all requirements, the current choices are:

- Ice 3.6 (non-optional)
- Java 11
- Nginx 1.14 or higher
- PostgreSQL 11

Other options

The installation walkthroughs provided in the documentation try to stick to a minimum installation. The only requirements are an understanding of the Unix shell, the standard package manager for your platform, and the regular Python distribution mechanisms.

However, more advanced installation mechanisms are available if you are interested and have familiarity with the given mechanism:

- *Ansible roles* are available for most installation steps. The primary roles, *omero-server* and *omero-web* have not yet been released and will need to be installed from GitHub.
- A *conda channel* provides pre-built packages needed by OMERO if you prefer to use Anaconda/Miniconda instead of the Python distribution provided by your platform.
- *Docker images* are also available. Both the *omero-server* and *omero-web* images are considered production quality.

Please get in touch at <https://forum.image.sc/c/data> if you have any questions.

OMERO.server

The steps for an OMERO.server installation have not changed substantially.

Download the OMERO.server.zip as you would usually do, and unpack it under your installation directory. We suggest `/opt/omero/server/` and symlink the unpacked directory to *OMERO.server*

We highly recommend a virtualenv-based installation for all of the Python dependencies. Follow the *standard installation instructions* for your platform. All instructions use a virtual environment.

Once you have your installation in place, you will need to follow the *standard upgrade instructions*, working from a *copy* of your data.

OMERO.web

Although it is possible to also follow the previous installation steps for OMERO.web, installation no longer requires downloading a package from <https://downloads.openmicroscopy.org>. If you choose to follow this newly introduced route, all requirements will be installed directly into the virtualenv for OMERO.web. Instructions are available under *web-deployment*.

Note that setting of OMERODIR variable is now required to specify where the OMERO installation lives. This defines where configuration files and log files will be stored. We suggest `/opt/omero/web` as the root for your installation.

The *upgrade guide* can help you to transfer your previous configuration. Moving forward, however, web upgrades should be much simpler under Python 3. Only a *pip install -U* of the appropriate libraries should be necessary.

Plugins

Core OMERO.web plugins have been updated for Python 3 and released to PyPI e.g.

```
pip install 'omero-iviewer>=0.9.0'
```

2.3.3 OMERO.server upgrade

The OME team is committed to providing frequent, project-wide upgrades both with bug fixes and new functionality. We try to make the schedule for these releases as public as possible. You may want to take a look at the [Trello boards](#) for exactly what will go into a release. See also *OMERO.web upgrade*.

See the full details of OMERO 5.6.10 features in the *CHANGELOGS*.

This guide aims to be as definitive as possible so please do not be put off by the level of detail; upgrading should be a straightforward process.

Warning: If you are upgrading from a version *prior to* OMERO 5.5 then you *must* also study the upgrade instructions for those prior versions because they may describe important steps that these instructions assume to already have been done by OMERO 5.5 users. Before proceeding with these instructions you may first need to read the [instructions](#) for upgrading *to* OMERO 5.5 because some extra steps may be required beyond simply running the SQL upgrade scripts described below.

Upgrade checklist

- *Check prerequisites*
- *File limits*
- *Password usage*
- *Memoization files invalidation*
- *Troubleshooting*
- *Upgrade check*

Check prerequisites

Before starting the upgrade, please ensure that you have reviewed and satisfied all the *system requirements* with *correct versions* for installation. In particular, ensure that you are running a suitable version of PostgreSQL to enable successful upgrading of the database, otherwise the upgrade script aborts with a message saying that your database server version is less than the OMERO prerequisite.

File limits

You may wish to review the open file limits. Please consult the *Too many open file descriptors* section for further details.

Password usage

The passwords and logins used here are examples. Please consult the *Which user account and password do I use where?* section for explanation. In particular, be sure to replace the values of **db_user** and **omero_database** with the actual database user and database name for your installation.

Memoization files invalidation

All cached Bio-Formats memoization files created at import time will be invalidated by the server upgrade. This means the very first loading of each image after upgrade will be slower. After re-initialization, a new memoization file will be automatically generated and OMERO will be able to load images in a performant manner again.

These files are stored under `BioFormatsCache` in the OMERO data directory, e.g. `/OMERO/BioFormatsCache`. You may see error messages in your log files when an old memoization file is found; to avoid these messages delete everything under this directory before starting the upgraded server.

It is possible to regenerate the memoization files before the user loads an image for the first time. For more information, read [MemoFileRegenerationReadMe.md](#).

Troubleshooting

If you encounter errors during an OMERO upgrade, database upgrade, etc., you should retain as much log information as possible and notify the OMERO.server team via the [forum](#).

Upgrade check

All OMERO products check themselves with the OmeroRegistry for update notifications on startup. If you wish to disable this functionality you should do so now as outlined on the [OMERO upgrade checks](#) page.

Upgrade steps

For all users, the basic workflow for upgrading your OMERO.server is listed below. Please refer to each section for additional details.

- *Check ahead for upgrade issues*
- *Perform a database backup*
- *Copy new binaries*
- *Upgrade your database*
- *Merge script changes*
- *Update your environment variables and memory settings*
- *Dependencies*
- *Server certificate*
- *Restart your server*
- *Restore a database backup*

Check ahead for upgrade issues

There is a **precheck** SQL script provided that performs various database checks to verify readiness for upgrade. The precheck script works even with the OMERO server running so it may be used before downtime for the actual upgrade is scheduled. Issues that the script reports will need to be resolved before the upgrade may proceed. The precheck script will **not** make any changes to the database: it merely performs various precautionary checks also done by the actual upgrade script.

```
$ cd OMERO.server
$ psql -h localhost -U db_user omero_database < sql/psql/OMERO5.4__0/OMERO5.3__
↳ 1-precheck.sql
Password for user db_user:
...
...
                                status
-----
                                +
                                +
                                +
```

```
YOUR DATABASE IS READY FOR UPGRADE TO VERSION OMERO5.4__0
```

```
+
```

```
+
```

```
+
```

```
(1 row)
```

Warning: The `sql/psql/OMERO5.4__0/OMERO5.3__1-precheck.sql` script referenced by the above `psql` command assumes a planned upgrade from OMERO 5.3.4. If you are instead currently running OMERO 5.3.3 or an earlier 5.3.x version then you perform the precheck by using the above command with `sql/psql/OMERO5.4__0/OMERO5.3__0-precheck.sql`. That script verifies that the database contains no trace of <https://www.openmicroscopy.org/security/advisories/2017-SV5-filename-2> having been exploited; this vulnerability was fixed in OMERO 5.3.4.

Perform a database backup

The first thing to do before **any** upgrade activity is to backup your database.

```
$ pg_dump -h localhost -U db_user -Fc -f before_upgrade.db.dump omero_database
```

Copy new binaries

Before copying the new binaries, stop the existing server:

```
$ cd Omero.server
$ omero admin stop
```

Your OMERO configuration is stored using `config.xml` in the `etc/grid` directory under your `OMERO.server` directory. Assuming you have not made any file changes within your `OMERO.server` distribution directory, you are safe to follow the following upgrade procedure:

```
$ cd ..
$ mv Omero.server Omero.server-old
$ unzip Omero.server-5.6.10-ice36.zip
$ ln -s Omero.server-||version_openmicroscopy|-ice36 Omero.server
$ cp Omero.server-old/etc/grid/config.xml Omero.server/etc/grid
```

Upgrade your database

Warning: This section only concerns users upgrading from a 5.3 or earlier server. If upgrading from a 5.4 or 5.5 server, you do not need to upgrade the database.

Ensure Unicode character encoding

OMERO requires a Unicode-encoded database; without it, the upgrade script aborts with a message warning how the OMERO database character encoding must be UTF8. From `psql`:

```
# SELECT datname, pg_encoding_to_char(encoding) FROM pg_database;
datname | pg_encoding_to_char
```

(continues on next page)

(continued from previous page)

```

-----+-----
template1 | UTF8
template0 | UTF8
postgres  | UTF8
omero     | UTF8
(4 rows)

```

Alternatively, simply run **psql -l** and check the output. If your OMERO database is not Unicode-encoded with UTF8 then it must be re-encoded.

If you have the **pg_upgradecluster** command available then its **--locale** option can effect the change in encoding. Otherwise, create a Unicode-encoded dump of your database: dump it *as before* but to a different dump file and with an additional **-E UTF8** option. Then, create a Unicode-encoded database for OMERO and restore that dump into it with **pg_restore**, similarly to *effecting a rollback*. If required to achieve this, the **-E UTF8** option is accepted by both **initdb** and **createdb**.

Run the upgrade script

You **must** use the same username and password you have defined during *OMERO.server installation*. For a large production system you should plan for the fact that the upgrade script may take several hours to run.

```

$ cd OMERO.server
$ psql -h localhost -U db_user omero_database < sql/psql/OMERO5.4__0/OMERO5.3__1.sql
Password for user db_user:
...
...

```

```

                                status
-----+-----
                                +
                                +
                                +
YOU HAVE SUCCESSFULLY UPGRADED YOUR DATABASE TO VERSION OMERO5.4__0 +
                                +
                                +

```

(1 row)

If you are upgrading from a server earlier than 5.3, then you must run the earlier upgrade scripts in sequence before the one above. There is no need to download and run the server from an intermediate major release but you must still study the upgrade instructions for earlier versions in case there are additional steps. For example, any optional SQL scripts that affect the database probably run only on the specific version before the next upgrade script.

Note: If you perform the database upgrade using *SQL shell*, make sure you are connected to the database using **db_user** before running the script. See [this forum thread](#) for more information.

Warning: The `sql/psql/OMERO5.4__0/OMERO5.3__1.sql` script referenced by the above **psql** command assumes upgrade from OMERO 5.3.4. If you are instead currently running OMERO 5.3.3 or an earlier 5.3.x version then you upgrade the database directly to OMERO 5.4.0 by using the above command with `sql/psql/OMERO5.4__0/OMERO5.3__0.sql`.

Optimize an upgraded database (optional)

After you have run the upgrade script, you may want to optimize your database which can both save disk space and speed up access times.

```
$ psql -h localhost -U db_user omero_database -c 'VACUUM FULL VERBOSE ANALYZE;'
```

Merge script changes

If any new official scripts have been added under `lib/scripts` or if you have modified any of the existing ones, then you will need to backup your modifications. Doing this, however, is not as simple as copying the directory over since the core developers will have also improved these scripts.

For further information on managing your scripts, refer to *OMERO.scripts*. If you require help, please contact the OME developers.

Update your environment variables and memory settings

Environment variables

If you changed the directory name where the 5.6.10 server code resides, make sure to update any system environment variables. Before restarting the server, make sure your `PATH` system environment variable is pointing to the new location. Also make sure the `OMERODIR` environment variable is set to the location of the server.

See *Environment variables* for more information.

JVM memory settings

Your memory settings should be copied along with `etc/grid/config.xml`, but you can check the current settings by running `omero admin jvmcfig`. See *Memory configuration* for more information.

Dependencies

While upgrading the server you should keep `OMERO.py` dependencies up to date to ensure that security updates are applied:

```
$ # first, activate virtualenv where omero-py is installed. Then upgrade:
$ pip install --upgrade 'omero-py>=5.19.0'
```

Server certificate

The server should be configured with at least a self-signed certificate to allow clients to establish secure connections.

Since OMERO 5.6.2, the recommended way to ensure that all OMERO server installations have at minimum, a self-signed certificate is to use the `omero-certificates` plugin. The plugin will generate or update your self-signed certificates and configure the OMERO.server. For the configuration to take effect, the server needs to be restarted. If you prefer to configure the OMERO server certificate manually, check *Client Server SSL verification*.

If your server has been configured with a version of `omero-certificates` older than 0.3.0 or manually, the configuration may need to be upgraded in particular to disallow the deprecated TLS 1.0 and 1.1 protocols.

To do so, activate the virtual environment where the server Python dependencies are installed, upgrade `omero-certificates` to version 0.3.0 or later, remove the `omero.glacier2.IceSSL.Protocols` and `omero.glacier2.IceSSL.ProtocolVersionMax` configurations and finally re-execute the **omero certificates** command:

```
$ pip install "omero-certificates>=0.3"
$ omero config set omero.glacier2.IceSSL.Protocols
$ omero config set omero.glacier2.IceSSL.ProtocolVersionMax
$ omero certificates
```

Note: From version 0.3.0, the **omero certificates** command adds TLS 1.3 to the list of TLS protocols allowed assuming the OMERO.server environment supports the protocol. In order to negotiate this protocol, clients will also need to be upgraded to depend on `omero-blitz 5.7.0` or greater (Java) or `omero-py 5.15.0` or greater (Python).

Restart your server

- Following a successful database upgrade, you can start the server.

```
$ omero admin start
```

- If anything goes wrong, please send the output of **omero admin diagnostics** to the [forum](#).

Restore a database backup

If the upgraded database or the new server version do not work for you, or you otherwise need to rollback to a previous database backup, you may want to restore a database backup. To do so, create a new database,

```
$ createdb -h localhost -U postgres -E UTF8 -O db_user omero_from_backup
```

restore the previous archive into this new database,

```
$ pg_restore -Fc -d omero_from_backup before_upgrade.db.dump
```

and configure your server to use it.

```
$ omero config set omero.db.name omero_from_backup
```

2.3.4 OMERO.web upgrade

The OME team is committed to providing frequent, project-wide upgrades with security fixes, bug fixes and new functionality. We try to make the schedule for these releases as public as possible. You may want to take a look at the [Trello boards](#) for exactly what will go into a release. See also *OMERO.server upgrade*.

See the full details of OMERO 5.6.10 features in the *CHANGELOGS*.

This guide aims to be as definitive as possible so please do not be put off by the level of detail; upgrading should be a straightforward process.

Upgrade checklist

- *Check prerequisites*
- *Upgrade*
- *Configuration*
- *Plugin updates*
- *Clear the sessions store (optional)*
- *Restart OMERO.web*
- *Troubleshooting*
- *Maintenance & Scaling*

Check prerequisites

Before starting the upgrade, please ensure that you have reviewed and satisfied all the *system requirements* with *correct versions* for installation.

Upgrade

Make sure you have activated the correct virtual environment then upgrade OMERO.web via pip:

```
$ pip install --upgrade 'omero-web>=5.24.0'
```

If the `omero-web` upgrade *requires* an upgrade to `omero-py` (e.g. for new features), this will happen automatically above. However, even when an `omero-py` upgrade is not required, there may be some benefits to upgrading:

```
$ pip install --upgrade 'omero-py>=5.19.0'
```

Configuration

We now recommend that `omero-web` is installed in a separate python virtual environment.

If you are migrating to a new virtual environment, where `$OMERODIR` does not refer to a server with an existing config, you will need to export and re-import the configuration from your previous installation.

```
OLD_INSTALLATION/bin/omero config get --show-password > properties.backup
# omero-web virtual env
omero config load properties.backup
```

If you generated configuration stanzas using **omero web config** which enables OMERO.web via NGINX, you should regenerate your config files, remembering to merge in any of your own modifications if necessary. You should carry out this step even for minor version upgrades as there may be fixes which require it.

```
omero web config nginx > new.conf
```

More examples can be found under *Configuration*.

Plugin updates

OMERO.web plugins are very closely integrated into the webclient. For this reason, it is possible that an update of OMERO will cause issues with an older version of a plugin. It is best when updating the server to also install any available plugin updates according to their own documentation.

All official OMERO.web plugins can be installed from [PyPI](#). You should remove all previously installed plugins and install the latest versions using `pip`.

Clear the sessions store (optional)

During an upgrade, it might be necessary to clear the sessions store before restarting OMERO.web. This is the case when either OMERO.web or Django introduced backwards incompatible changes in the way sessions are stored or if the session serialization format has been modified via `omero.web.session_serializer`.

Warning: Clearing the sessions store will terminate all OMERO.web sessions and log out all users.

The process for clearing the session store depends on the storage backend:

- OMERO.web sessions are stored on the filesystem if the `omero.web.session_engine` property is set to either `omero.web.filesessionstore` or `django.contrib.sessions.backends.file` or if `omero.web.session_engine` is unset.

Sessions are stored under a temporary folder determined by the output of `tempfile.gettempdir()`, usually `/tmp`. By default, each session is stored as a separate file prefixed with `sessionid` so the following command will delete all stored sessions:

```
$ rm /tmp/sessionid*
```

If `omero.web.session_cookie_name` is set, its value will be used as the prefix of the file sessions and the command above needs to be modified accordingly.

- OMERO.web sessions are stored using the [Redis](#) store if the `omero.web.session_engine` property is set to `django.contrib.sessions.backends.cache` and Redis is configured via the `omero.web.caches` property by setting the `BACKEND` to `django.core.cache.backends.redis.RedisCache` and `LOCATION` to the URL of the Redis instance.

Sessions are stored as key/value pairs in the Redis store with the name of the key including `django.contrib.sessions.cache` and can be managed using the **redis-cli** utility. Assuming a default local Redis store, the following command can be used to delete all the Redis keys associated with OMERO.web sessions:

```
$ redis-cli keys '*django.contrib.sessions.cache*' | xargs redis-cli del
```

If Redis URL points to a different hostname, port or database number, the command above needs to be adjusted accordingly.

Restart OMERO.web

Finally, restart OMERO.web with the following command:

```
$ omero web restart
```

Troubleshooting

If you encounter errors during an OMERO.web upgrade, etc., you should retain as much log information as possible, including the output of **omero web diagnostics** to the OMERO team via the mailing lists available on the [support](#) page.

Maintenance & Scaling

If you have not already done so, there are a number of additional steps that can be performed on your OMERO.web installation to improve its functioning. For example, you may need to set up a regular task to clear out any stale OMERO.web session files. More information can be found in the various walkthroughs available from [OMERO.web installation and maintenance](#).

Additionally, it is recommended to use a WSGI-capable server such as NGINX. Information can be found under [OMERO.web installation and maintenance](#).

2.4 Maintenance

This section contains instructions for administering, troubleshooting, and backing-up your installation.

2.4.1 Troubleshooting OMERO

Which user account and password do I use where?

Accounts table, including the example usernames and passwords used in the installation guides:

Account type	Function	Username	Password
System	Administrator/Root		
System	(Database) service account	postgres	
System	(OMERO) service account	omero_user	
Database	Database administrator	postgres	
Database	Database user	db_user	db_password
OMERO	OMERO administrator	root	root_password
OMERO	OMERO users		

Note: These example usernames and passwords are provided to help you follow the installation guide examples. Do not use root_password or db_password; substitute your own passwords.

There are a total of three types of user accounts: system, database and OMERO accounts.

System accounts

These are accounts on your machine or directory server (e.g. LDAP, Active Directory). One account is used for running the OMERO server (either your own or one you created specially for running OMERO, referred to here as “omero_user”). There is also a user called the “root-level user” on the [installation page](#). A separate “postgres” user is used for running the database server. The “omero_user” account runs the OMERO server, and owns the files uploaded to OMERO. This account must have permission to write to the */OMERO/* binary repository. Some operations in the install scripts require the root-level/administrator-level privileges in order to use another account to perform particular actions e.g. the “postgres” user to create a database. However **the OMERO.server should never be run as the root-level/administrator-level user or as the system-level “postgres” user.**

Database accounts

The PostgreSQL database system contains user and administrative accounts; these are completely separate from the system accounts, above, existing only inside the database. The database administrative user (“postgres”) is the owner of all the database resources, and can create new users internal to the database. A single database account is used at run time by OMERO to talk to your database. Therefore, you must configure the [database](#) values during installation:

```
$ omero config set omero.db.user db_user
$ omero config set omero.db.pass db_password
```

Note: Do **not** use db_user or db_password here; substitute your own username and password.

A database user may have the same name as an account on your machine, in which case a password might not be necessary.

OMERO accounts

These accounts only exist inside the OMERO system, and are completely separate from both the system and database accounts, above. The first user which you will need to configure is the “root” OMERO user (different from any root-level Unix account). This is done by setting the password in the database script, see [Database tools](#).

Other OMERO users can be created via the OMERO.web admin tool. None of the passwords have to be the same, in fact they should be different **unless you are using the LDAP plugin**.

Server fails to start

1. Check that you are able to successfully connect to your PostgreSQL installation as outlined on the [PostgreSQL page](#)).
2. Check the permissions on your `omero.data.dir` (`/OMERO` by default) as outlined on the [OMERO.server installation](#) page.
3. Are you on a laptop? If you see an error message mentioning “`node master couldn’t be reached`”, you may be suffering from a network address swap. Ice does not like to have its network changed as can happen if the server is running on a laptop on wireless. If you lose connectivity to icegridnode, you may have to kill it manually via `kill PID` or `killall icegridnode` (under Unix).
4. If you see an error message mentioning “`Freeze::DatabaseException`” or “`could not lock file: var/registry/__Freeze/lock`”, your icegrid registry may have become corrupted. This is not a problem, but it will be necessary to stop OMERO and delete the `var/master` directory (e.g. `rm -rf var/master`). When restarting OMERO, the registry will be automatically re-created.

5. If you see an error message mentioning “Protocol family unavailable”, you will need to set the `Ice.IPv6` property with **`omero config set Ice.IPv6 0`**.
6. If you upgraded from a 5.0.2 server or older and copied the entire content of the `etc/grid` directory from the old server to the new server, you will need to revert the changes made to `templates.xml` to [generate the new memory settings](#).
7. If OMERO starts up, but fails to respond to connection requests, check `netstat -a` for port 4064. If nothing is listening on 4064, you may need to specify which network interface to use. **`omero config set Ice.Default.Host 127.0.0.1`**, for example, would force OMERO to only listen on localhost. See [Proxy and Endpoint Syntax](#) for more information.

Remote clients cannot connect to OMERO installation

OMERO.web connects but not OMERO.insight

The Admin section of OMERO.web appears to work properly and you may or may not have created some users, but no matter what you do remote clients will not speak to OMERO. OMERO.insight gives you an error message similar to the following despite giving the correct username and password:



This is often due to firewall misconfiguration on the machine that runs your OMERO server, affecting the ability of remote clients to locate it. A common issue is when port TCP/4064 and/or TCP/4063 is not opened, run `telnet server-name 4064` (resp. 4063) to check if this is the case. The output of the command should be:

```
Trying server-name...
Connected to server-name.
Escape character is '^['
```

Please see the [Server security and firewalls](#) page for more information.

SSL connection issues

- `javax.net.ssl.SSLHandshakeException: DH ServerKeyExchange does not comply to algorithm constraints`
- `javax.net.ssl.SSLHandshakeException: The server selected protocol version TLS10 is not accepted by client preferences [TLS12]`
- `SSL handshake failure: The parameter is incorrect.`
- `reason = SSL error occurred for new outgoing connection: remote address = XXX.XXX.XXX.XXX:4064 dh key too small`

These errors indicate the client is unable to establish a secure connection with the server. Deployment platforms show a trend of making the transport layer security policy tighter by default.

The recommended way to overcome SSL connection issues for OMERO clients connecting to the server is to use the [omero-certificates](#) plugin available from [PyPI](#).

Follow the instructions from [Server certificate](#) to create and configure self-signed certificates as necessary on the OMERO.server and restart it as normal for the changes to take effect.

Server crashes with...

- X11 connection rejected because of wrong authentication
- X connection to localhost:10.0 broken (explicit kill or server shutdown).

OMERO uses image scaling and processing techniques that may be interfered with when used with SSH (Secure Shell) X11-forwarding. You should disable SSH X11-forwarding in your SSH session by using the `-x` flag as follows before you restart the OMERO.server:

```
ssh -x my_server.examples.com
```

OutOfMemoryError / PermGen space errors in OMERO.server logs

Out of memory or permanent generation (PermGen) errors can be caused by many things. You may be asking too much of the server. Or you may require an increase in the maximum Java heap or the permanent generation space. This can be done by modifying the configuration for your OMERO.server. See [Memory configuration](#).

Similarly, if you are finding out of memory errors in one of the other process logs (e.g. `Indexer-0.log` or `PixelData-0.log`), you might try optimizing the JVM memory settings.

Furthermore, under certain conditions access of images greater than 4GB can be problematic on 32-bit platforms due to certain bugs within the Java Virtual Machine including [Bug ID: 4724038](#). A 64-bit platform for your OMERO.server is **HIGHLY** recommended.

Too many open files

This is most often seen as an error during importing and is caused by the number of opened files exceeding the limit imposed by your operating system. It might be due to OMERO leaking file descriptors; if you are not using the latest version, please upgrade, since a number of bugs which could cause this behavior have been fixed. It is also possible for buggy scripts which do not properly release resources to cause this error.

To view the current per-process limit, run

```
ulimit -Hn
```

which will show the hard limit for the maximum number of file descriptors (`-Sn` will show the soft limit). This limit may be increased. On Linux, see `/etc/security/limits.conf` (global PAM per-user limits configuration); it is also possible to increase the limit in the shell with

```
ulimit -n newlimit
```

providing that you are uid 0 (other users can only increase the soft limit up to the hard limit). To view the system limit, run

```
cat /proc/sys/fs/file-max
```


We recommend 8K as a minimum number of files limit for production systems, with 16K being reasonable for bigger machines.

On Mac OS X, the standard ulimit will not work properly. There are several different ways of setting the ulimit, depending upon the version of OS X you are using, but the most common are to edit `sysctl.conf` or `launchd.conf` to raise the limit. However, note that both of these methods change the defaults for every process on the system, not just for a single user or service.

Increasing the number of available filehandles via ‘ulimit -n’

ValueError: filedescriptor out of range in select() - this is a known issue in Python versions prior to 2.7.0. See [#6201](#) and Python [Issue #3392](#) for more details.

Directory exists but is not registered

Import errors of type `Directory exists but is not registered: CheckedPath(username_id)` suggest a server-side issue under the `ManagedRepository`.

For production servers, this can be caused by a server crash during import or an issue at the file system level (permissions, renaming). If the `ManagedRepository/username_id` folder is empty, you should try removing it before trying another import.

For development servers, this may be caused by the binary directory not being cleaned after the database has been wiped.

See also:

Upload problem: `Directory exists but is not registered`.

import: `Directory exists but is not registered: CheckedPath(`

`[ome-devel] Directory exists but is not registered: CheckedPath(username_id)`

Not enough heap space

```
java.lang.OutOfMemoryError: Java heap space
```

If you get an out of memory error, you can try increasing the maximum Java heap space, by setting the `JAVA_OPTS` variable before running the import command. For example to set a maximum heap space of 3GB:

```
$ export JAVA_OPTS=-Xmx3G
$ omero import ...
```

Another change that may be required is to adjust the `OMERO.server` configuration. Run the following command:

```
$ omero config set omero.jvmcfg.percent 22 # 15 is the default
```

Then restart the `OMERO.server`.

DropBox fails to start: failed to get session

If the main server starts but DropBox fails with the following entry in `var/log/DropBox.log`,

```
2011-06-07 03:42:56,775 ERROR [      fsclient.DropBox] (MainThread) Failed to get_
↪Session:
```

then it may be that the server is taking a relatively long time to start.

A solution to this is to increase the number of retries and/or the period (seconds) between retries in `etc/grid/templates.xml`

```
<property name="omero.fs.maxRetries" value="5"/>
<property name="omero.fs.retryInterval" value="3"/>
```

Search does not return expected results

If searching for a specific term does not return the expected results (e.g. searching for the name of a tag does not return the full list of a user's images annotated with that tag), there are a few things that may have gone wrong. See [Missing search results](#) for more details.

OMERO.web issues

OMERO.web running but status says not started

If you upgraded OMERO but forgot to stop OMERO.web, processes will still be running. In order to kill stale processes by hand, run:

```
$ ps aux | grep django.pid
```

Note: As Gunicorn is based on the pre-fork worker model it is enough to kill the master process, the one with the lowest PID.

OMERO.web not available HTTP 404

Consult `nginx.error.log` for more details.

The most common problem appears when the default configuration for `location /` is loaded prior to `omeroweb.conf`

```
2016/01/01 00:00:00 [error] 1234#0: *5 "/usr/share/nginx/html/webclient/login/index.html
↪" is not found (2: No such file or directory), client: 1.2.3.4, server
```

OMERO.web not responding/timeout issues

```
[CRITICAL] WORKER TIMEOUT (pid:1234)
```

OMERO.web deployed with Gunicorn relies on the operating system to provide all of the load balancing while handling requests. Adjust the timeout using `omero.web.wsgi_timeout` and scale the number of `omero.web.wsgi_workers` starting with $(2 \times \text{NUM_CORES}) + 1$ workers. For more details refer to [Configuration](#).

Issues with downloading data from OMERO.web

An [Configuration](#) is available for testing with nginx if you are encountering problems with downloads failing. You can also configure OMERO.web to limit downloads - refer to the [OMERO.web installation and maintenance](#) documentation and [Download restrictions](#) for further details.

OMERO.web piecharts

‘Drive space’ does not generate pie chart or ‘My account’ does not show markup picture and crop the picture options.

Error message says: ‘Piechart could not be displayed. Please check log file to solve the problem’. Please check `var/log/OMEROweb.log` for more details. There are a few known possibilities:

- ‘TclError: no display name and no \$DISPLAY environment variable’. Turn off the compilation of TCL support in [Matplotlib](#).
- ‘ImportError: No module named Image’. Install [Pillow](#) (packages should be available for your distribution). Also double check if all of the prerequisites were installed from [OMERO.web deployment](#).

OMERO.web fails to start

If OMERO.web fails to start either with an error or type `binascii.Error: Incorrect padding` or `json.decoder.JSONDecodeError`, your existing sessions are likely incompatible and you will need to follow the steps indicated at [Clear the sessions store \(optional\)](#) to clear the sessions store.

Troubleshooting performance issues with the clients

If you are having issues with slowdown and timeouts in the clients, there are three things to check:

- your network connection
- if you are running out of memory (processing large datasets can cause problems)
- whether your server’s HOME directory is on a network share

In the final case, two issues arise. Firstly, when performing some operations, the clients make use of temporary file storage and log directories, as described in the [Client configuration](#) section of [System requirements](#). If your home directory is stored on a network, possibly NFS mounted (or similar), then these temporary files are being written and read over the network which can slow access down. Secondly, the OMERO.server also accesses the temporary and log folders of the user the server process is running as. As the server makes heavier use of these folders than the clients, if the OMERO.server user directory is stored on a network drive then slow performance can occur.

To resolve this, define the `OMERO_TMPDIR` environment variable to point at a temporary directory located on the local file system (e.g. `/tmp/omero`).

Other issues

Connection problems and TCP window scaling on Linux systems

Later versions of the 2.6 Linux kernel, specifically 2.6.17, have TCP window scaling enabled by default. If you are having initial logins never timeout or problems with connectivity in general you can try turning the feature off as follows:

```
# echo 0 > /proc/sys/net/ipv4/tcp_window_scaling
```

Server or clients print “WARNING: Prefs file removed in background...”

```
Nov 12, 2008 3:02:50 PM java.util.prefs.FileSystemPreferences$7 run
WARNING: Prefs file removed in background /root/.java/.userPrefs/prefs.xml
Nov 12, 2008 3:02:50 PM java.util.prefs.FileSystemPreferences$7 run
WARNING: Prefs file removed in background /usr/lib/jvm/java-1.7.0-icedtea-1.7.0.0/jre/.
↪systemPrefs/prefs.xml
```

These warnings (also sometimes listed as ERRORS) can be safely ignored, and are solely related to how Java is installed on your system. See [Bug ID: 4751177](#) or this [ome-users thread](#) on our mailing list for more information.

Data corruption

If you are dealing with a data corruption issue, you may find the information on [PixelService resolution order for locating binary data for images](#) useful.

PyTables version

Note that PyTables has dropped support for Python 3.8 in the 3.9.x line, see [tag v3.9.1](#) .

2.4.2 OMERO.cli as an OMERO admin tool

When first beginning to work with the OMERO server, the **omero db**, **omero config**, and **omero admin** commands will be the first you will need. For other important uses of Command Line see the links in “See Also” box.

Database tools

Rather than try to provide the functionality of a RDBM tool like `psql`, the **omero db script** command helps to generate SQL scripts for building your database. You can then use those scripts from whatever tool is most comfortable for you:

```
$ omero db script --password secretpassword
```

```
Using OMERO5.4 for version
Using 0 for patch
Using password from commandline
Saving to /home/omero/OMERO5.4__0.sql
```

If you do not specify the OMERO root password on the command line you will be prompted to enter it.

Server configuration

The **omero config** command is responsible for reading/writing user-specific profiles stored under \$OMERODIR/etc/grid/config.xml. To get the current profile, use the **omero config def** command:

```
$ omero config def
default
```

You can then examine the current profile keys using **omero config get** and set key-value pairs using **omero config set**:

```
$ omero config get

$ omero config set example "my first value"

$ omero config get
example=my first value
```

You can use the Omero_CONFIG environment variable to point at a different profile, e.g.:

```
$ Omero_CONFIG=another omero config def
another

$ Omero_CONFIG=another omero config get

$ Omero_CONFIG=another omero config set example "my second value"

$ Omero_CONFIG=another omero config get
example=my second value
```

The values set via **omero config set** override those compiled into the server jars. The default values which are set can be seen in [Configuration properties glossary](#). To add several values to a configuration, you can pipe them via standard in using **omero config load**. To grep for the example LDAP configuration from [omero-server.properties](#)

```
$ grep omero.ldap src/main/resources/omero-server.properties | Omero_CONFIG=ldap omero_
↪config load

$ Omero_CONFIG=ldap omero config get
omero.ldap.attributes=objectClass
omero.ldap.base=ou=example,o=com
omero.ldap.config=false
omero.ldap.groups=
omero.ldap.keyStore=
omero.ldap.keyStorePassword=
omero.ldap.new_user_group=default
omero.ldap.password=
omero.ldap.protocol=
omero.ldap.trustStore=
omero.ldap.trustStorePassword=
omero.ldap.urls=ldap://localhost:389
omero.ldap.username=
omero.ldap.values=person
```

Each of these values can then be modified to suit your local setup. To remove one of the key-value pairs, pass no second argument:

```
$ OMERO_CONFIG=ldap omero config set omero.ldap.trustStore
$ OMERO_CONFIG=ldap omero config set omero.ldap.trustStorePassword
$ OMERO_CONFIG=ldap omero config set omero.ldap.keyStore
$ OMERO_CONFIG=ldap omero config set omero.ldap.keyStorePassword

$ OMERO_CONFIG=ldap omero config get
omero.ldap.attributes=objectClass
omero.ldap.base=ou=example,o=com
omero.ldap.config=false
omero.ldap.groups=
omero.ldap.new_user_group=default
omero.ldap.password=
omero.ldap.protocol=
omero.ldap.urls=ldap://localhost:389
omero.ldap.username=
omero.ldap.values=person
```

If you will be using a particular profile more frequently you can set it as your default using the **omero config def** command:

```
$ omero config def ldap
```

And finally, if you would like to remove a profile, for example to wipe a given password off of a system, use **omero config drop**:

```
$ omero config drop
```

Server administration

Server start

Once your database has been properly configured and your config profile is set to use that database, you are ready to start your server using the **omero admin start** command:

```
$ omero admin start
```

This command performs the following operations in order:

1. rewrites the configuration files if *omero admin start --force-rewrite* is passed or the server has never been started
2. checks the server status, i.e. pings the **master** node using the IceGrid administration tool
3. aborts the command if a server is running
4. rewrites the configuration files if it has not been done at step 1
5. starts the server
6. waits until the server is up

Most configuration files under `etc/grid` are generated using the templates under `etc/grid/templates` and the server configuration stored in `etc/grid/config.xml`. The rewriting step updates the JVM memory settings (see [Memory configuration](#)) and the server ports (see [Ports](#)) based on the server configuration.

-h, --help

Display the help for this subcommand.

--foreground

This option is particularly useful for debugging and maintenance and allows for starting the server up in the foreground, that is without creating a daemon on UNIX-like systems. During the lifetime of the server, the prompt from which it was launched will be blocked.

--force-rewrite

This option forces the server configuration files under `etc/grid` to be rewritten before the status of the server is checked.

Server stop

To stop a running server, you can invoke the **omero admin stop** subcommand:

```
$ omero admin stop
```

This command does the following operations in order:

1. rewrites the server configuration files if `omero admin stop --force-rewrite` is passed
2. checks the server status, i.e. pings the **master** node using the IceGrid administration tool
3. aborts the command if no server is running
4. stops the server
5. waits until the server is down

-h, --help

Display the help for this subcommand.

--force-rewrite

This option forces the configuration files to be rewritten before the server status is checked.

Server restart

To stop and start the server in a single command, you can use the **omero admin restart** command:

```
$ omero admin restart
```

The `restart` subcommand supports the same options as **omero admin start**.

Server diagnostics

To debug a server or inspect the configuration, you can use the **omero admin diagnostics** command:

```
$ omero admin diagnostics
```

The output of this command will report information about:

- the server prerequisites (**psql**, **java**)
- the server environment variables
- the server memory settings and ports
- the status of the binary repository

User/group management

The **omero user** and **omero group** commands provide functionalities to add and manage users and groups on your database.

See also:

- *Moving objects between groups*
- *Changing ownership of objects*

User creation

New users can be added to the database using the **omero user add** command:

```
$ omero user add -h
```

During the addition of the new user, you will need to specify the first and last name of the new user and their username as well as the groups the user belongs to. To add John Smith identified as `jsmith` as a member of the group named `test-group`, enter:

```
$ omero user add jsmith John Smith --group-name test-group
```

Additional parameters such as the email address, institution, middle name etc. can be passed as optional arguments to the **omero user add** command.

For managing the permissions of restricted administrators, *OMERO.cli does provide means* but that functionality is not yet offered in a friendly manner by the **omero user** command. The *OMERO.web Admin interface* is recommended for this task instead.

If you are using ldap as an authentication backend, you can create an OMERO user account for `jsmith` using the **omero ldap create** command, which allows the administrator to add `jsmith` to an OMERO group, before they have ever logged in to OMERO:

```
$ omero ldap create jsmith
```


Converting non-LDAP users to LDAP authentication

If you want to take an existing (non-LDAP) user and ‘upgrade’ them to using LDAP you can do so using the **omero ldap setdn** command:

```
$ omero ldap setdn -h
```

The process is also reversible so that the OMERO password for a user rather than the LDAP password will be used. See the caveat in the setdn help output below:

```
usage: omero ldap setdn [-h] [--user-id USER_ID]
                        [--user-name USER_NAME]
                        [--group-id GROUP_ID]
                        [--group-name GROUP_NAME] [-C]
                        [-s SERVER] [-p PORT] [-g GROUP]
                        [-u USER] [-w PASSWORD] [-k KEY]
                        [--sudo ADMINUSER] [-q]
                        choice
```

Enable or disable LDAP login for user (admins only)

Once LDAP login is enabled for a user, the password set via OMERO is ignored, and any attempt to change it will result in an error. When you disable LDAP login, the previous password will be in effect, but if the user never had a password, one will need to be set!

Positional Arguments:

choice	Enable/disable LDAP login (true/false)
--------	--

Optional Arguments:

In addition to any higher level options

-h, --help	show this help message and exit
--user-id USER_ID	ID of the user.
--user-name USER_NAME	Name of the user.
--group-id GROUP_ID	ID of the group.
--group-name GROUP_NAME	Name of the group.

Login arguments:

Environment variables:

OMERO_USERDIR	Set the base directory containing the user's files. Default: \$HOME/omero
OMERO_SESSIONDIR	Set the base directory containing local sessions. Default: \$OMERO_USERDIR/sessions
OMERO_TMPDIR	Set the base directory containing temporary files. Default: \$OMERO_USERDIR/tmp
OMERO_PASSWORD	Set the user's password for creating new sessions. Ignored if -w or --password is used.

Optional session arguments:

(continues on next page)

(continued from previous page)

-C, --create	Create a new session regardless of existing ones
-s SERVER, --server SERVER	OMERO server hostname
-p PORT, --port PORT	OMERO server port
-g GROUP, --group GROUP	OMERO server default group
-u USER, --user USER	OMERO username
-w PASSWORD, --password PASSWORD	OMERO password
-k KEY, --key KEY	OMERO session key (UUID of an active session)
--sudo ADMINUSER	Create session as this admin. Changes meaning of ↵
↵password!	
-q, --quiet	Quiet mode. Causes most warning and diagnostic ↵
↵messages to be suppressed.	

User deactivation

To deactivate a user, remove him/her from the system group user. Use the command **omero user leavegroup** and specify the user group as the target:

```
# Remove jsmith from group user
$ omero user leavegroup user --name=jsmith
```

To reactivate the user, add him/her back to the system group user i.e.:

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

User editing

Updating the details of a user e.g. the email address can be achieved using the **omero obj update** command:

```
# Determine the ID of jsmith
$ omero user info --user-name jsmith
# Change the email address of jsmith. Replace 123 by the ID of jsmith
$ omero obj update Experimenter:123 email=jsmith@new_address.com
```

Group creation

New groups can be added to the database using the **omero group add** command:

```
$ omero group add -h
```

During the addition of the new group, you need to specify the name of the new group:

```
$ omero group add newgroup
```

The permissions of the group are set to *private* by default. Alternatively you can specify the permissions using **--perms** or **--type** optional arguments:

```
$ omero group add read-only-1 --perms='rwr---'
$ omero group add read-annotate-1 --type=read-annotate
```

See also:

Groups and permissions system

Description of the group permissions levels.

Lists of users/groups on the OMERO server can be queried using the **omero user list** and **omero group list** commands:

```
$ omero user list
$ omero group list
```

Group membership

Users can be added to existing groups using the **omero user joingroup** or **omero group adduser** commands. Similarly, users can be removed from existing groups using the **omero user leavegroup** or **omero group removeuser** commands:

```
# Add jsmith to group read-annotate-1
$ omero group adduser jsmith --name=read-annotate-1
# Remove jsmith from group read-annotate-1
$ omero group removeuser jsmith --name=read-annotate-1
# Add jsmith to group read-only-1
$ omero user joingroup read-only-1 --name=jsmith
# Remove jsmith from group read-only-1
$ omero user leavegroup read-only-1 --name=jsmith
```

By passing the **--as-owner** option, these commands can also be used to manage group owners

```
# Add jsmith to the owner list of group read-annotate-1
$ omero group adduser jsmith --name=read-annotate-1 --as-owner
# Remove jsmith from the owner list of group read-annotate-1
$ omero user leavegroup read-annotate-1 --name=jsmith --as-owner
```

Group copy

To create a copy of a group, you must first create a new group using the **omero group add** command:

```
$ omero group add read-only-2 --perms='rwr---'
```

Then you can use the **omero group copyusers** command to copy all group members from one group to another:

```
$ omero group copyusers read-only-1 read-only-2
```

To copy the group owners, use the same command with the **--as-owner** optional argument:

```
$ omero group copyusers read-only-1 read-only-2 --as-owner
```

Group modification

To change the permissions of a group, for example to make the group `read-annotate-1` a read-write group, run:

```
$ omero group perms --perms='rwrw--' --name='read-annotate-1'
```

If you want to change its name to `read-write-1` afterwards, run:

```
$ omero obj update ExperimenterGroup:123 name='read-write-1'
```

Adjusting administrator restrictions

OMERO 5.4 introduced the concept of a *restricted administrator*. The meaning and representation of the server's underlying permissions restrictions is described in *developer documentation*. OMERO.web offers easy management of restrictions and is recommended for setting up restricted administrators via its [Admin tab](#).

OMERO.cli does not offer easy management of restrictions because support is yet to be added. In the meantime it can already manipulate administrator restrictions in the awkward manner described hereunder.

Warning: OMERO.web provides a simplified view of the available restrictions: the *permissions mapping* is such that checking *one* box in the web interface may lift *multiple* underlying restrictions from an administrator. The recommended OMERO.web management interface may thus prove confusing if OMERO.cli has been used to set a combination of restrictions that does not correspond to those bundles of related restrictions available in OMERO.web.

View an administrator's restrictions

For an administrator with user ID 123,

```
$ omero hql "SELECT ap.name FROM Experimenter user JOIN user.config AS ap WHERE user.id_
↳ = 123 AND ap.name LIKE 'AdminPrivilege:%' AND LOWER(ap.value) <> 'true' ORDER BY ap.
↳ name"
```

lists their applicable restrictions such that the administrator may *not* exercise privileges for that operation.

Set a restriction on an administrator

For an administrator with user ID 123,

```
$ omero obj map-set Experimenter:123 config AdminPrivilege:SomePrivilege false
```

restricts them so that they may no longer exercise *SomePrivilege*.

Clear a restriction from an administrator

For an administrator with user ID 123,

```
$ omero obj map-set Experimenter:123 config AdminPrivilege:SomePrivilege true
```

removes a restriction so that they may exercise *SomePrivilege*.

Note: You may not clear a restriction from an administrator if you have that same restriction applying to yourself.

Warning: Never clear *AdminPrivilege:ReadSession* from a restricted administrator unless clearing *all* their restrictions to make them into a full administrator. No restricted administrator should be able to read all OMERO sessions.

Repository management

Since 5.0.3 it is possible to list images, filesets and the repositories that contain them. At an administrator-only level it is also possible to move filesets. This functionality is provided by the **omero fs** command. See

```
$ omero fs -h
```

Listing repositories

The **omero fs repos** subcommand lists the repositories used by OMERO. For example

```
omero fs repos
```

#	Id	UUID	Type	Path
0	1	83bf5c68-8236-47ff-ae3e-728674eb0103	Managed	/OMERO/ManagedRepository
1	2	ad899754-bff0-4605-a234-acd4da178f3b	Public	/OMERO
2	3	ScriptRepo	Script	/dist/lib/scripts

The options available to this subcommand are:

-h, --help

Display the help for this subcommand.

--style {plain,csv,json,sql}

This option determines the output style, tabular *sql* being the default as in the previous example. The *csv* style is comma-separated values with an initial header row, *plain* is the same as *csv* but without the header row. *json* returns an array of JSON objects that can be piped to other tools.

--managed

This option lists only Managed repositories.

For example

```
omero fs repos --managed --style=csv
```

```
#,Id,UUID,Type,Path
```

```
0,1,83bf5c68-8236-47ff-ae3e-728674eb0103,Managed,/OMERO/ManagedRepository
```

Listing filesets

The **omero fs sets** subcommand lists filesets by various criteria. Filesets are bundles of original data imported into OMERO 5 and above, which represent one or more images. For example

```
omero fs sets
```

#	Id	Prefix	Images	Files	Transfer
0	79853	user-3_9/2014-07/22/16-41-04.244/	1	1	
1	79852	user-3_9/2014-07/22/10-44-11.235/	1	1	
2	79851	user-3_9/2014-07/22/10-44-07.300/	1	1	
3	79813	user-3_9/2014-07/21/14-13-02.353/	1	1	
4	79812	user-3_9/2014-07/21/14-13-00.182/	1	1	
5	79811	user-3_9/2014-07/21/14-12-59.212/	1	1	
6	79810	user-3_9/2014-07/21/14-12-57.896/	1	1	
7	79809	user-3_9/2014-07/21/14-10-22.436/	3	600	
...					
24	79772	user-4_5/2014-07/18/17-14-43.631/	1	1	

(25 rows, starting at 0 of approx. 50173)

The options available to this subcommand are:

-h, --help

Display the help for this subcommand.

--style {plain,csv,json,sql}

See *omero fs repos --style*.

--limit LIMIT

This option specifies the maximum number of return values, the default is 25.

--offset OFFSET

This option specifies the number of entries to skip before starting the listing, the default, 0, is to skip no entries.

--order {newest,oldest,prefix}

This option determines the order of the rows returned, *newest* is the default.

--without-images

This option lists only those filesets without images, these may be corrupted filesets.

--with-transfer WITH_TRANSFER [WITH_TRANSFER ...]

This option lists only those filesets imported using the given in-place import methods.

--check

This option checks each fileset for validity by recalculating each file's checksum and comparing it with the checksum recorded upon import. This may be slow. **This option is available to administrators only.**

--extended

With this option more details are provided for each returned value. This may be slow.

For example

```
omero fs sets --order oldest --limit 3 --offset 5 --check
```

#	Id	Prefix	Images	Files	Transfer	Check
0	54	user-3_9/2014-06/09/09-24-28.037/	1	1		OK
1	55	user-3_9/2014-06/09/09-24-31.354/	1	1		OK
2	57	user-5_4/2014-06/09/11-01-00.557/	1	1		OK

(3 rows, starting at 5 of approx. 78415)

Listing images

The **omero fs images** subcommand lists imported images by various criteria. This subcommand is useful for showing pre-FS (i.e. OMERO 4.4 and before) images which have their original data archived with them. For example

```
omero fs images
```

#	Image	Name	FS	# Files	Size
0	102803	kidney_Tf1_1.bmp.ome.tiff	79853	1	435.1 KB
1	102802	4kx4k.jpg	79852	1	1.7 MB
2	102801	2kx2k.jpg	79851	1	486.3 KB
3	102773	multi-channel.ome.tif	79813	1	220.3 KB
4	102772	multi-channel-z-series.ome.tif	79812	1	1.1 MB
5	102771	multi-channel-time-series.ome.tif	79811	1	1.5 MB
6	102770	multi-channel-4D-series.ome.tif	79810	1	7.4 MB
7	102769	001_001_000_000.tif [Well B6]	79809	600	1.1 GB
...					
24	102732	00027841.png	79774	1	235 B

(25 rows, starting at 0 of approx. 117393)

The options available to this subcommand are:

-h, --help

Display the help for this subcommand.

--style {plain,csv,json,sql}

See *omero fs repos --style*.

--limit LIMIT

See *omero fs sets --limit*.

--offset OFFSET

See *omero fs sets --offset*.

--order {newest,oldest,prefix}

See *omero fs sets --order*.

--archived

With this option the subcommand lists only images with archived data.

--extended

With this option more details are provided for each returned value. This may be slow.

For example

```
omero fs images --archived --offset 16 --limit 3
```

#	Image	Name	FS	# Files	Size
0	15481	UMD001_ORO.svs [Series 1]		1	12.7 MB
1	15478	biosamplefullframetif.tif		1	32.0 MB
2	10018	050118.lei [07-13-a]		4	4.8 MB

(3 rows, starting at 16 of approx. 833)

Renaming filesets

The **omero fs rename** subcommand moves an existing fileset, specified by its ID, to a new location. **This subcommand is available to administrators only.**

It may be useful to rename an existing fileset after the import template (`omero.fs.repo.path`) has been changed to match the new template. By default the files in the fileset and the accompanying import log are moved. For example, after adding the group name and group ID to template and changing the date format

```
$ omero fs rename 9
```

Renaming Fileset:9 to pg-0_3/user-0_2/2014-07-23/16-48-20.786/
Moving user-0_2/2014-07/23/16-31-51.070/ to pg-0_3/user-0_2/2014-07-23/16-48-20.786/
Moving user-0_2/2014-07/23/16-31-51.070.log to pg-0_3/user-0_2/2014-07-23/16-48-20.786.
↪ log

The ID can be given as a number or in the form *Fileset:ID*.

The options available to this subcommand are:

-h, --help

Display the help for this subcommand.

--no-move

With this option the files will be left in place to be moved later. Advice will be given as to which files need to be moved to complete the renaming process. Note that if the files are not moved then the renamed filesets will not be accessible until the files have been moved into the new positions.

For example

```
$ omero fs rename Fileset:8 --no-move
```

Renaming Fileset:8 to pg-0_3/user-0_2/2014-07-23/16-49-23.543/
Done. You will now need to move these files manually:

```
-----
mv /OMERO/ManagedRepository/user-0_2/2014-07/23/16-29-14.809/ /OMERO/ManagedRepository/
↪pg-0_3/user-0_2/2014-07-23/16-49-23.543/
mv /OMERO/ManagedRepository/user-0_2/2014-07/23/16-29-14.809.log /OMERO/
↪ManagedRepository/pg-0_3/user-0_2/2014-07-23/16-49-23.543.log
```

Note: The **omero fs rename** subcommand is currently disabled pending a bug-fix.

Detailing disk usage

The **omero fs usage** subcommand provides details of the underlying disk usage for various types of objects. This subcommand takes optional positional arguments of object types with ids and returns the total disk usage of the specified objects.

For example

```
omero fs usage Image:30001,30051 Plate:1051 --report
```

Total disk usage: 1064320138 bytes in 436 files

component	size (bytes)	files
Thumbnail	582030	256
Job	1772525	2
Pixels	49545216	12
FilesetEntry	1011947729	124
Annotation	472638	42

(5 rows)

If no positional argument is given then the total usage for the current user across all of that user's groups is returned.

For example

```
omero fs usage --report
```

Total disk usage: 4526436430274 bytes in 26078 files

component	size (bytes)	files
Pixels	14654902013	2961
FilesetEntry	4510839804505	8820
Thumbnail	17337131	8110
Job	265665153	2792
OriginalFile	1757277	109
Annotation	13167582976	3910

(6 rows)

If multiple objects are given and those objects contain common data then that usage will not be counted twice. For example, if two datasets contain the same image then the fileset for that image will not be double-counted in the total disk usage.

The options available to this subcommand are:

-h, --help

Display the help for this subcommand.

--style {plain,csv,json,sql}

See *omero fs repos --style*.

--wait WAIT

Number of seconds to wait for the processing to complete. To wait indefinitely use < 0, for no wait use 0. The default is to wait indefinitely.

--size_only

Print total bytes used, in bytes, with no extra text, this is useful for automated scripting.

--report

Print detailed breakdown of disk usage by types of files. This option is ignored if `--size_only` is used.

--units {K,M,G,T,P}

Units to use for disk usage for the total size using base-2. The default is bytes.

--groups

Print size for all of the current user's groups, this includes the user's own data and the data of other group members visible to the user. This option only applies if no positional arguments are given.

For example

```
omero fs usage --groups --size_only -C -u user-1

4576108188820

omero fs usage Project:1,2 Dataset:5 --units M --report

Total disk usage: 1432 MiB in 121 files
component | size (bytes) | files
-----+-----+-----
Thumbnail | 73710         | 34
Pixels    | 1499341282    | 34
Annotation | 3000028       | 53
(3 rows)
```

Creating directories

For directory creation in a Managed repository use **omero fs mkdir**: this creates both the directory on the underlying filesystem and the corresponding entry in the OMERO server's database. The new directory will be owned by the root user and in the user group. The options available to this subcommand are:

-h, --help

Display the help for this subcommand.

--parents

Ensure that the whole given path exists in the Managed repository. Analogous to the common **mkdir**'s `--parents` option, originally simply `-p` in IEEE Std 1003.1-2008.

See also:

Command Line Interface as an OMERO client

User documentation for the Command Line Interface

Command Line Interface as an OMERO development tool

Developer Documentation for the Command Line Interface

Help for any specific CLI command can be displayed using the `-h` argument. See [Command line help](#) for more information.

2.4.3 OMERO.server backup and restore

Cleaning up your binary repository

As detailed in *Binary data*, it is possible that some files may be left behind when a delete action is performed. This was mostly an issue on Windows, which is no longer supported for OMERO server, but is still possible on Posix systems. If you think files have been left behind e.g. after a hard-reboot, a script to clean these up is included in the OMERO.server distribution `lib/python/omero/util/cleanse.py`, which can be used so:

```
$ omero admin cleanse /OMERO
```

Note that only items not listed in the relational database (i.e. previously failed deletes) and empty directories will be cleaned up by this script.

Note: If you are cleaning a large repository and the process runs for a long time but does not appear to succeed, you may find that running `$ omero sessions keepalive` in one shell and then running the `cleanse` command from another shell allows the process to finish without timing out.

Managing OMERO.server log files

Your OMERO.server will produce log files that are rotated when they reach 512MB. These directories will look like:

```
omero_dist $ ls var/log
Blitz-0.log      FileServer.log      MonitorServer.log   Processor-0.log     master.out
DropBox.log     Indexer-0.log       OMEROweb.log        master.err
```

Any files with a `.1`, `.2`, `.3` etc. suffix may be compressed or deleted.

OMERO.server log file location

The log file directory may also be relocated to different storage by modifying the `etc/grid/default.xml` file:

```
...
<variable name="OMERO_LOGS"    value="var/log/" />
...
```

Backing up OMERO

Understanding backup sources

OMERO.server has three main backup sources:

1. PostgreSQL database (assumed to be `omero_database`)
2. OMERO.server *binary data store* (assumed to be `/OMERO`)
3. OMERO.server configuration

Warning: You must back up (1) and (2) frequently.

Frequent backups taken while the server is still running are usually sufficient but you should be aware that they may not be consistent snapshots. The **safest** course of action is to perform backups during server downtime when possible, especially if you think you may need the backup.

You need to back up (3) only before you make changes. You can copy it into `/OMERO/backup` to ensure it is kept safe:

```
$ omero config get > /OMERO/backup/omero.config
```

Other backup sources

If you have edited `etc/grid/(win)default.xml` directly for any reason then you will also need to copy that file to somewhere safe, such as `/OMERO/backup`.

The `lib/scripts` directory should also be backed up, but restoring it may pose issues if any of your users have added their own “official scripts”. A github repository is available at <https://github.com/ome/scripts> which provides help for merging your scripts directories.

Backing up your PostgreSQL database

Database backups can be achieved using the PostgreSQL `pg_dump` command. Here is an example backup script that can be placed in `/etc/cron.daily` to perform daily database backups:

```
#!/bin/bash

DATE=`date '+%Y-%m-%d_%H:%M:%S-%Z'`
OUTPUT_DIRECTORY=/OMERO/backup/database
DATABASE="omero_database"
DATABASE_ADMIN="postgres"

mkdir -p $OUTPUT_DIRECTORY
chown -R $DATABASE_ADMIN $OUTPUT_DIRECTORY
su $DATABASE_ADMIN -c "pg_dump -Fc -f $OUTPUT_DIRECTORY/$DATABASE.$DATE.pg_dump $DATABASE"
↪ "
```

Other database backup configurations are outside the scope of this document but can be researched on the [PostgreSQL website](#) (*Chapter 25. Backup and Restore*).

Note: Frequent backups of your PostgreSQL database are crucial; you do not want to be in the position of trying to restore your server without one.

Note: Consider Omero database dumps to be sensitive and be accordingly cautious in allowing access to them. For example, the `session.uuid` column contains UUIDs with which Omero clients can attach to existing sessions.

Backing up your binary data store

To simplify backup locations we have, in this document, located all database and configuration backups under `/OMERO`, your *binary data store*. The entire contents of `/OMERO` should be backed up frequently as this will, especially if this document's conventions are followed, contain all the relevant data to restore your OMERO.server installation in the unlikely event of a system failure, botched upgrade or user malice.

File system backup is often a very personal and controversial topic amongst systems administrators and as such the OMERO project does not make any explicit recommendations about backup software. In the interest of providing a working example we will use open source `rdiff-backup` project and like *Backing up your PostgreSQL database* above, provide a backup script which can be placed in `/etc/cron.daily` to perform daily `/OMERO` backups:

```
#!/sh
#!/bin/bash

FROM=/OMERO
TO=/mnt/backup_server

rdiff-backup $FROM $TO
```

`rdiff-backup` can also be used to backup `/OMERO` to a remote machine:

```
#!/sh
#!/bin/bash

FROM=/OMERO
TO=backup_server.example.com:~/backup/omero

rdiff-backup $FROM $TO
```

More advanced `rdiff-backup` configurations are beyond the scope of this document. If you want to know more you are encouraged to read the documentation available on the `rdiff-backup` [website](#).

Restoring OMERO

There are three main steps to OMERO.server restoration in the event of a system failure:

1. OMERO.server etc configuration
2. PostgreSQL database (assumed to be `omero`)
3. OMERO.server binary data store (assumed to be `/OMERO`)

Note: It is important that restoration steps are done in this order unless you are absolutely sure what you are doing.

Restoring your configuration

Once you have retrieved an OMERO.server package from the [downloads](#) page that **matches** the version you originally had installed, all that is required is to restore your backup preferences by running:

```
$ omero config load /OMERO/backup/omero.config
```

You should then follow the *Reconfiguration* steps of [install](#).

Restoring your PostgreSQL database

If you have had a PostgreSQL crash and database users are missing from your configuration, you should follow the first two (*Create a non-superuser database user* and *Create a database for OMERO data to reside in*) steps of [OMERO.server installation](#). Once you have ensured that the database user and empty database exist, you can restore the pg_dump file as follows:

```
$ sudo -u postgres pg_restore -Fc -d omero_database omero.2010-06-05_16:27:29-GMT.pg_dump
```

Restoring your OMERO.server binary data store

All that remains once you have restored your Java preferences and PostgreSQL database is to restore your */OMERO binary data store* backup.

See also:

List of backup software

Wikipedia page listing the backup softwares.

PostgreSQL 10 Interactive Manual

Chapter 25: Backup and Restore

rdiff-backup documentation

Online documentation of rdiff-backup project

2.4.4 OMERO upgrade checks

On each startup the OMERO server checks for available upgrades via the [UpgradeCheck](#) class. An HTTP GET call is made to the URL configured in [omero-common.properties](#) as `omero.upgrades.url`, currently `http://upgrade.openmicroscopy.org.uk` by default (note that viewing that link in your browser will redirect you to this page).

Note: If you have been redirected here by clicking on a link to `http://upgrade.openmicroscopy.org.uk` in an error message or log while trying to run one of the **Bio-Formats command line tools**, please see the [Bio-Formats command line tools documentation](#) for assistance.

Actions

Currently the only action taken when an upgrade is necessary is a log statement at WARN level.

```
2017-09-01 12:21:32,070 WARN [ome.system.UpgradeCheck] (main) ↪
UPGRADE AVAILABLE: Please upgrade to 5.6.10 See https://downloads.openmicroscopy.org/
latest/omero for the latest version
```

Future versions may also send emails and/or IMs to administrators. In the case of critical upgrades, the server may refuse to start.

Privacy

Currently, the only information which is being transmitted to the server is:

- Java virtual machine version
- operating system details (architecture, version and name)
- current server version
- poll frequency (for determining statistics)
- your IP address (standard HTTP header information)

Note: Currently the “poll” property is unused.

If this is a problem for your site, please see *Disabling* below.

Disabling

If you would prefer to have no checks made, the check can be disabled by setting the `omero.upgrades.url` property to an empty string:

```
omero.upgrades.url=
```

Developers

To use the `UpgradeCheck` class from your own code, it is necessary to have `omero-common-x.y.z.jar` on your classpath. Then,

```
String version = "yourAppVersion" // e.g. 5.5.0;
ResourceBundle bundle = ResourceBundle.getBundle("omero-common");
String url = bundle.getString("omero.upgrades.url");
ome.system.UpgradeCheck check = new UpgradeCheck(
    url, version, "insight"); // Or "importer", etc.
check.run();
check.isUpgradeNeeded();
// optionally
check.isExceptionThrown();
```

will connect to the server and check your current version against the latest release.

See also:

OMERO.server installation

Instructions for installing OMERO.server on UNIX and UNIX-like platforms

OMERO.server upgrade

Instructions for upgrading OMERO.server

Server security and firewalls

Description of OMERO security practices

2.4.5 Moving the data repository

It may be necessary to move either the whole OMERO data directory or only the Managed Repository to a new location on the file system. This should be done with care following the steps below.

Warning: Before moving OMERO data it is wise to ensure that both the data and the database are fully backed up. See [OMERO.server backup and restore](#).

The current location of the data repositories can be found using the **fs repos** command:

```
$ omero fs repos
```

#	Id	UUID	Type	Path
0	1	309ea513-a23c-48d1-abd2-9ceed1b3ffa4	Managed	/Users/omero/var/omero/ManagedRepository
1	2	ScriptRepo	Script	/Users/omero/dist/lib/scripts
2	3	3ec8c878-c776-48a3-b57e-2a11b0c97045	Public	/Users/omero/var/omero

(3 rows)

Note: This command can be slow, **omero config get** can also be used to determine if `omero.data.dir` or `omero.managed.dir` have non-default values.

Moving the OMERO data directory

If the Managed Repository is within the OMERO data directory and the whole data directory is to be moved then the following steps should be used:

```
omero admin stop
omero config set omero.data.dir NEW
mv OLD NEW
omero admin start
```

Warning: The use of **omero config set** is absolutely necessary here. The steps: **omero admin stop**, **mv**, **omero admin start** without **omero config set** could lead to an unstable situation.

For example, moving the OMERO data directory from `/Users/omero/var/omero` to `/Volumes/omero`:


```
$ omero admin stop
...
$ omero config set omero.data.dir /Volumes/omero
$ mv /Users/omero/var/omero /Volumes/omero
$ omero admin start
...
$ omero fs repos
```

#	Id	UUID	Type	Path
0	1	309ea513-a23c-48d1-abd2-9ceed1b3ffa4	Managed	/Volumes/omero/ManagedRepository
1	2		ScriptRepo	/Users/omero/dist/lib/scripts
2	3	3ec8c878-c776-48a3-b57e-2a11b0c97045	Public	/Volumes/omero

(3 rows)

Moving the Managed Repository

If the Managed Repository is in a separate location from the OMERO data directory or only the Managed Repository is to be moved then the following steps should be used:

```
omero admin stop
omero config set omero.managed.dir NEW
mv OLD NEW
omero admin start
```

Warning: The use of **omero config set** is absolutely necessary here. The steps: **omero admin stop**, **mv**, **omero admin start** without **omero config set** could lead to an unstable situation.

For example, moving the Managed Repository from `/Users/omero/var/omero/ManagedRepository` to `/Volumes/imports/ManagedRepository`:

```
$ omero admin stop
...
$ omero config set omero.managed.dir /Volumes/imports/ManagedRepository
$ mv /Users/omero/var/omero/ManagedRepository /Volumes/imports/ManagedRepository
$ omero admin start
...
$ omero fs repos
```

#	Id	UUID	Type	Path
0	1	309ea513-a23c-48d1-abd2-9ceed1b3ffa4	Managed	/Volumes/imports/ManagedRepository
1	2		ScriptRepo	/Users/omero/dist/lib/scripts
2	3	3ec8c878-c776-48a3-b57e-2a11b0c97045	Public	/Users/omero/var/omero

(3 rows)

Note: If `omero.managed.dir` is not set then the location of the Managed Repository will be determined by `omero.data.dir` and the OMERO directory should only be moved as a whole.

If the Managed Repository needs to be moved to a location other than that set by `omero.data.dir`, to a location outside of the OMERO data directory, for example, then `omero.managed.dir` must be set.

If `omero.managed.dir` is set then the Managed Repository and the OMERO data directory should be treated independently and thus be moved separately if necessary.

Extending the Managed Repository

It is possible to leave the Managed Repository in place yet have newly imported image files stored on a different underlying storage volume. For example, if your `omero.managed.dir` is set to `/mnt/omero/ManagedRepository` then, as that volume fills, it would become better for new imports to be stored elsewhere. An OMERO administrator may use the **omero fs mkdir** subcommand to properly set up a subdirectory for that new volume in the existing Managed Repository:

```
omero fs mkdir volume-B
```

This is the correct way to create `/mnt/omero/ManagedRepository/volume-B` ready for new imports. The new storage volume may then be mounted at that mount point. Alternatively, if the volume is already mounted elsewhere, such as `/mnt/omero/large-volume-B/`, then while the OMERO server is shut down you may create a corresponding symbolic link at `/mnt/omero/ManagedRepository/volume-B`:

```
rmkdir /mnt/omero/ManagedRepository/volume-B
ln -s /mnt/omero/large-volume-B /mnt/omero/ManagedRepository/volume-B
```

In either case the `omero.fs.repo.path` must be updated in the server configuration. An example of adjusting its usual default value is:

```
omero config set omero.fs.repo.path 'volume-B/%user%_%userId%//%year%-%month%/%day%/
↪%time%'
```

After the OMERO server is started then new imports should upload onto the new storage volume. At a later date further storage volumes may be added by using this same workflow.

2.5 Optimizing Server Configuration

This section discusses the options for configuring OMERO.server for optimum performance and security.

2.5.1 Server security and firewalls

General

OMERO has been built with security in mind. Various standard security practices have been adhered to during the development of the server and client including:

- Encryption of all passwords between client and server via SSL
- Full encryption of all data when requested via SSL
- User and group based access control

- Authentication via LDAP
- Limited visible TCP ports to ease firewalling
- Use of a higher level language (Java or Python) to limit buffer overflows and other security issues associated with native code
- Escaping and bind variable use in all SQL interactions performed via Hibernate

Note: The OMERO team treats the security of all components with care and attention. If you have a security issue to report, please do not hesitate to contact us using our private, secure mailing list as described on the [Security](#) page.

Firewall configuration

Securing your OMERO system with so called *firewalling* or *packet filtering* can be done quite easily. By default, OMERO clients only need to connect to two TCP ports for communication with your OMERO.server: 4063 (unsecured) and 4064 (SSL). These are the [IANA](#) assigned ports for the Glacier2 router from [ZeroC](#). Both of these values, however, are completely up to you, see [SSL](#) below.

Important OMERO ports:

- **TCP/4063**
- **TCP/4064**

If you are using OMERO.web, then you will also need to make your HTTP and HTTPS ports available. These are usually 80 and 443.

Important OMERO.web ports:

- **TCP/80**
- **TCP/443**

Example OpenBSD firewall rules

```
block in log on $ext_if from any to <omero_server_ip>
pass in on $ext_if proto tcp from any to <omero_server_ip> port 4063
pass in on $ext_if proto tcp from any to <omero_server_ip> port 4064
pass in on $ext_if proto tcp from any to <omero_server_ip> port 443
pass in on $ext_if proto tcp from any to <omero_server_ip> port 80
```

Example Linux firewall rules

```
iptables -P INPUT drop
iptables -A INPUT -p tcp --dport 4063 -j ACCEPT
iptables -A INPUT -p tcp --dport 4064 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
...
```

Passwords

The passwords stored in the `password` table are salted and hashed, so it is impossible to recover a lost one, instead a new one must be set by an admin.

If the password for the root user is lost, the only way to reset it (in the absence of other admin accounts) is to manually update the password table. The **omero** command can generate the required SQL statement for you:

```
$ omero db password
Please enter password for OMER0 root user:
Please re-enter password for OMER0 root user:
UPDATE password SET hash = 'PJueOtwuTPHB8Nq/1rFVxg==' WHERE experimenter_id = 0;
```

Current hashed password:

```
$ psql mydatabase -c " select * from password"
experimenter_id |          hash
-----+-----
              0 | Xr4il0zQ4PC0q3aQ0qbuaQ==
(1 row)
```

Change the password using the generated SQL statement:

```
$ psql mydatabase -c "UPDATE password SET hash = 'PJueOtwuTPHB8Nq/1rFVxg==' WHERE_
↪experimenter_id = 0;"
UPDATE 1
```

Stored data

The server's binary repository and database contain information that may be confidential. Afford access only on a limited and necessary basis. For example, the *ReadSession warning* is for naught if the restricted administrator can read the contents of the `session` table.

Java key- and truststores

If your server is connecting to another server over SSL, you may need to configure a truststore and/or a keystore for the Java process. This happens, for example, when your LDAP server uses SSL. See the *LDAP plugin* for information on how to configure the LDAP URLs. As with all configuration properties, you will need to restart your server after changing them.

To do this, you will need to configure several server properties, similar to the properties you configured during *installation*.

- truststore path

```
omero config set omero.security.trustStore /home/user/.keystore
```

A truststore **is** a database of trusted entities **and** their associated X.509 certificate chains authenticating the corresponding public keys. The truststore contains the Certificate Authority (CA) certificates **and** the certificate(s) of the other party to which this entity intends to send encrypted (confidential) data. This file must contain the public key certificates of the CA **and** the client's public key certificate.

If you don't have one you can create it using the following:

```
openssl s_client -connect {{host}}:{{port}} -prexit < /dev/null | openssl x509 -
↳outform PEM | keytool -import -alias ldap -storepass {{password}} -keystore {
↳{{truststore}} -noprompt
```

- truststore password

```
omero config set omero.security.trustStorePassword secret
```

- keystore path

```
omero config set omero.security.keyStore /home/user/.mystore
```

A keystore **is** a database of private keys **and** their associated X.509 certificate chains authenticating the corresponding public keys.

A keystore **is** mostly needed **if** you are doing client-side certificates **for** authentication against your LDAP server.

- keystore password

```
omero config set omero.security.keyStorePassword secret
```

SSL

Especially if you are going to use LDAP authentication to your server, it is important to encrypt the transport channel between clients and the Glacier2 router to keep your passwords safe.

By default, all logins to OMERO occur over SSL using an anonymous handshake. After the initial connection, communication is un-encrypted to speed up image loading. Clients can still request to have all communications encrypted by clicking on the lock symbol. An unlocked symbol means that non-password related activities (i.e. anything other than login and changing your password) will be unencrypted, and the only critical data which is passed in the clear is your session id.

Administrators can configure OMERO such that unencrypted connections are not allowed, and the user's choice will be silently ignored. The SSL and non-SSL ports are configured in the `etc/grid/default.xml` file and, as described above, default to 4064 and 4063 respectively and can be modified using the [Ports](#) configuration properties. For instance, to prefix all ports with 1, use `omero.ports.prefix`:

```
$ omero config set omero.ports.prefix 1
```

You can disable unencrypted connections by redirecting clients to the SSL port using the server property `omero.router.insecure`:

```
$ omero config set omero.router.insecure "OMERO.Glacier2/router:ssl -p 4064 -h @omero.
↳host@"
```

If you want to force host verification see [Client Server SSL verification](#).

See also:

[LDAP authentication](#)

2.5.2 LDAP authentication

LDAP is an open standard for querying and modifying directory services that is commonly used for authentication, authorization and accounting (AAA). OMERO.server supports the use of an LDAP server to query (but not modify) AAA information for the purposes of automatic user creation.

This allows OMERO users to be automatically created and placed in groups according to your existing institution policies. This can significantly simplify your user administration burden. Note that OMERO has its own concept of “groups” that is quite distinct from LDAP groups.

The OMERO.server LDAP implementation can handle a number of use cases. For example:

- Allow every “inetOrgPerson” under `omero.ldap.base` to login
- but restrict access based upon an arbitrary LDAP filter, e.g.

```
omero.ldap.user_filter=(memberOf=cn=someGoup,ou=Lab,o=College)
```

- and add that user to some number of groups, e.g.

```
omero.ldap.new_user_group=:query:(member=@{dn})
```

How it works

On login, the username provided is searched for in OMERO. If the name does not exist, then the LDAP plugin is queried for a username matching the system-wide user filter. If such an LDAP entry exists and the password matches, a new user with the given username is created, and the user is added to any groups which match the `new_user_group` setting.

On subsequent logins, the user filter and the password are again checked against the LDAP server, and if there is no longer a match, login is refused. If you would prefer to only have the `user_filter` applied during user creation and not on every login, see [Legacy password providers](#).

You can take existing non-LDAP users and ‘upgrade’ them to using LDAP with the OMERO command line tool, see [Converting non-LDAP users to LDAP authentication](#). You can also use **omero ldap create** to add an ldap user to OMERO groups without requiring them to log in first, see [User/group management](#) for details.

LDAP properties

The LDAP plugin is configured via several configuration properties, all starting with `omero.ldap` (see [LDAP](#)).

Some of the property values are passed directly to the underlying LDAP library ([Spring LDAP](#)), which in turn makes use of the Java API. OMERO does not modify the error messages thrown by the library or by Java, so please consult the appropriate documentation to diagnose any low-level problems.

Note: Please remember that once a change has been made, a server restart will be needed.

Minimum configuration

The following properties are the minimum requirements for logging in to OMERO using LDAP.

```
omero.ldap.config=true
omero.ldap.urls=ldap://localhost:389
omero.ldap.username=
omero.ldap.password=
omero.ldap.base=ou=example,o=com
```

After having configured your connection, you can turn LDAP on and off between restarts by setting `omero.ldap.config` to false. The base property determines where in the LDAP tree searches will begin. No users or groups will be found if they are not under the base provided.

User lookup

Two user properties are used to look up users by login name and, if necessary, create new users based on the information in LDAP.

```
omero.ldap.user_filter=(objectClass=person)
omero.ldap.user_mapping=omeName=cn,firstName=givenName,lastName=sn,email=mail,
↪institution=department,middleName=middleName
```

`omero.ldap.user_filter` will be AND'd to the username query, and can contain any valid LDAP filter string. The username query is taken from the LDAP attribute which gets mapped to “omeName” by `omero.ldap.user_mapping`. Here, the “cn” is mapped to “omeName”, so the username query is `(cn=[login name])`. The final query is `(&(objectClass=person)(cn=[login name]))`, which must return a single result to be considered valid.

Group lookup

Three group properties are all concerned with what groups a user will be placed in on creation.

```
omero.ldap.group_filter=(objectClass=groupOfNames)
omero.ldap.group_mapping=name=cn
omero.ldap.new_user_group=default
```

The group filter and group mapping work just as the user filter and mapping do, in that the group name query will be AND'd with the `group_filter`. In this case, the final query would be `(&(objectClass=groupOfNames)(cn=[group name]))`. However, these properties may not be used depending on the value of `new_user_group`, which can have several different values:

- If not prefixed at all, then the value is simply the name of a group which all users from LDAP should be added to.
- If prefixed with `:ou:`, then a user's last organizational unit (OU) will be used as his or her group. For example, the user with the DN “cn=frank,ou=TheLab,ou=LifeSciences,o=TheCollege” will be placed in the group “TheLab”.
- If prefixed with `:attribute:`, then the rest of the string is taken to be an attribute all of whose values will be taken as group names. For example, `omero.ldap.new_user_group=:attribute:memberOf` would add a user to all the groups named by `memberOf`. You can prefix this value with `filtered_` to have the `group_filter` applied to the attribute values, i.e. `:filtered_attribute:memberOf` will mean that only the values of `memberOf` which match `group_filter` will be considered. An example value of the `memberOf` attribute would be: `CN=mygroup,OU=My Group,OU=LabUsers, DC=openmicroscopy,DC=org`

- If prefixed with `:dn_attribute:`, then the rest of the string is taken to be an attribute all of whose values will be taken as group distinguished names. For example, `omero ldap.new_user_group=:dn_attribute:memberOf` would add a user to all the groups named by `memberOf`, where the name of the group is mapped via `group_mapping`. You can prefix this value with `filtered_` to have the `group_filter` applied to the attribute values, i.e. `:filtered_dn_attribute:memberOf` will mean that only the values of `memberOf` which match `group_filter` will be considered. An example value of the `memberOf` attribute would be: `CN=mygroup,OU=My Group,OU=LabUsers, DC=openmicroscopy,DC=org`

Note that if an attribute specified in `omero.ldap.group_mapping` does not constitute a part of the Distinguished Name (DN) as determined by your LDAP server then it can only be found by using `:attribute:` or `:filtered_attribute:` instead. Typical attributes that comprise the DN are: DC, CN, OU, O, STREET, L, ST, C and UID.

- If prefixed with `:query:`, then the rest of the value is taken as a query to be AND'ed to the group filter. In the query, values from the user such as `"@{cn}"`, `"@{email}"`, or `"@{dn}"` can be used as place holders.
- If prefixed with `:bean:`, then the rest of the string is the name of a Spring bean which implements the `NewUser-GroupBean` interface. See the developer documentation [LDAP plugin design](#) for more info.

Compound Filters

Note: OMERO uses standard [RFC 2254 LDAP filters](#), so they must conform to that syntax and are only able to do what those filters can do. You can test the filters via `ldapsearch` on your OMERO server (assuming you have the OpenLDAP binaries installed).

If you are using OpenLDAP make sure your directory has the `memberOf` attribute correctly configured. Some versions of ApacheDS do not support `memberOf` at all.

Both the `user_filter` and the `group_filter` can contain any valid LDAP filter string. These must be a valid filter in themselves. e.g.

```
omero.ldap.user_filter=(|(ou=Queensland Brain Institute)(ou=Ageing Dementia Research))
```

The “|” operator (read: “OR”) above allows members of two organizational units to login to OMERO. Expanding the list allows concentric “rings” of more and more OU’s granular access to OMERO.

```
omero.ldap.group_filter=(&(objectClass=groupOfNames)(mail=omero.flag))
```

The “&” operator (read: “AND”) produces a filter that will only match groups that have the `mail` attribute set to the value `omero.flag`. When combined with the `group_mapping`, the final query would be `(&(&(objectClass=groupOfNames)(mail=omero.flag))(cn=[group name]))`

This is the same as the query `(&(objectClass=groupOfNames)(mail=omero.flag)(cn=[group name]))` but setting `group_filter` to `(objectClass=groupOfNames)(mail=omero.flag)` is not valid as that is not a valid filter on its own.

To restrict the list of groups to just the ones returned by the above query, the following setting is also required to remove unmatched groups:

```
omero.ldap.new_user_group=:filtered_dn_attribute:memberOf
```


Case sensitivity

By default, the LDAP plugin is case-sensitive i.e. it will treat the usernames JSmith and jsmith as two different users. You can remove case sensitivity using:

```
omero config set omero.security.ignore_case true
```

Warning: Enabling this option will affect **all**, even non-LDAP, usernames in your OMERO system. It is the system administrator's responsibility to handle any username clashes which may result. Making non-LDAP usernames lowercase is required. Non-LDAP users with uppercase characters in their username will not be able to log in and will not appear in some administrative tools.

UPDATE experimenter SET omename = lower(omename); can be used on your database to make this change to all users if desired. This operation is irreversible.

LDAP over SSL

If you are connecting to your server over SSL, that is, if your URL is of the form `ldaps://ldap.example.com:636` you may need to configure a key and trust store for Java. See the [Server security and firewalls](#) page for more information.

Synchronizing LDAP on user login

This feature allows for LDAP to be considered the authority on user/group membership. With the following setting enabled, each time a user logs in to OMERO their LDAP groups will be read from the LDAP server and reflected in OMERO:

```
omero config set omero.ldap.sync_on_login true
```

Admin actions carried out in the clients may not survive this synchronization e.g. if an admin has removed an LDAP user from an LDAP group in the UI, the user will be re-added to the group when logging in again after the synchronization.

Note: This applies to groups created by LDAP in OMERO 5.1.x. Groups created in older versions of OMERO will not be registered as LDAP groups if you have manually altered their membership, even if the membership now matches the LDAP group.

omero ldap setdn true --group-name \$NAME can be used to make these previous OMERO groups into LDAP groups.

Legacy password providers

The primary component of the LDAP plugin is the `LdapPasswordProvider`, which is responsible for creating users, checking their passwords, and adding them to or removing them from groups. The default password provider is the `chainedPasswordProvider` which first checks LDAP if LDAP is enabled, and then checks JDBC. This can explicitly be enabled by executing the system admin command:

```
omero config set omero.security.password_provider chainedPasswordProvider
```

When the LDAP password provider implementation changes, previous versions can be configured as necessary.

- `chainedPasswordProviderNoSalt`

The `chainedPasswordProviderNoSalt` uses the version of the JDBC password provider without password salting support as available in the OMERO 4.4.x series. To enable it, use:

```
omero config set omero.security.password_provider chainedPasswordProviderNoSalt
```

- `chainedPasswordProvider431`

With the 431 password provider, the user filter is only checked on first login and not kept on subsequent logins. This allows for an OMERO admin to change the username of a user in omero to be different than the one kept in LDAP. To enable it, use:

```
omero config set omero.security.password_provider chainedPasswordProvider431
```

See also:

OMERO.server installation

Installation guide for OMERO.server under UNIX-based platforms

Server security and firewalls

Security pages for OMERO.server

LDAP plugin design

Developer documentation on extending the LDAP plugin yourself.

What are your LDAP requirements?

Forum discussion if you have LDAP requirements that are not covered by the above configuration

JNDI referrals documentation

<https://docs.oracle.com/javase/jndi/tutorial/ldap/referral/jndi.html>

Active Directory

Active Directory (AD) supports a form of LDAP and can be used by OMERO like most other directory services.

In AD, the [Domain Services](#) (DS) ‘forest’ is a complete instance of an Active Directory which contains one or more domains. Querying a particular Domain Service will yield results which are local to that domain only. In an environment with just one domain it is possible to use the default configuration instructions for OMERO LDAP. If there are multiple domains in the forest then it is necessary to query the [Global Catalogue](#) to enable querying across all of them.

Global Catalogue

In an AD DS forest, a [Global Catalogue](#) provides a central repository of all the domain information from all of the domains. This can be queried in the same way as a specific Domain Service using LDAP, but it runs on different ports; 3268 and 3269 (SSL).

- LDAP AD Global Catalogue server URL string

```
omero config set omero.ldap.urls ldap://ldap.example.com:3268
```

Note: A SSL URL above should look like this: `ldaps://ldap.example.com:3269`

2.5.3 Performance and monitoring

Once you have your OMERO server running and secured, a second critical step will be tuning various configuration parameters in order to get optimal performance. Assorted timeouts can be found under *Performance* but the more critical properties are outlined below.

Database configuration

The configuration properties starting with *omero.db* control how OMERO manages JDBC connections to your database. For a production system, `omero.db.poolsize` is the most important property to modify. By default, a limited number of simultaneous connections (e.g. 10) are allowed. You should plan for allowing a few connections *per concurrent user*.

```
$ omero config set omero.db.poolsize 100
```

Memory configuration

OMERO should automatically configure itself to take advantage of the physical memory installed on a system whilst leaving room for other services. You may wish to override the defaults on a production server, for instance if your machine is solely dedicated to running OMERO you can increase the amount of memory that OMERO will use. You may also need to modify your settings if you are seeing out-of-memory errors when dealing with certain types of images.

A number of configuration properties starting with *omero.jvmcfg* control the calculation of how much memory to allocate to various OMERO services on startup, most importantly:

- blitz
- indexer
- pixeldata

Configuration properties

Configuration properties can either be applied to all three service types at the same time by omitting the service type (e.g. `omero.jvmcfg.strategy`) or to each individually by including it (e.g. `omero.jvmcfg.strategy.blitz`).

For example, the default, `PercentStrategy`, is equivalent to making the call:

```
$ omero config set omero.jvmcfg.strategy percent
```

This could be changed to use the `ManualStrategy` for all servers:

```
$ omero config set omero.jvmcfg.strategy manual
```

Strategies

A couple of strategies are available for calculating the effective JVM settings from the provided configuration properties.

PercentStrategy

Default. Reads the *percent* configuration property which can also be set globally or on a service-type basis. This percentage (0-100) of the system memory is used for the process, subject to minimum and maximum limits which can be changed. `omero.jvmcfg.system_memory`, `omero.jvmcfg.min_system_memory`, and `omero.jvmcfg.max_system_memory` are all used for defining the system memory seen. The default percentages are: blitz and pixeldata 15%, indexer 10%. If `omero.jvmcfg.perm_gen` or `omero.jvmcfg.heap_size` are explicitly set, they will be used directly as with the *ManualStrategy*.

ManualStrategy

Simply provides the values given as the JVM settings. If no value is set for a particular configuration property, then the default is used: *heap_size=512m* and *perm_gen=128m*. These values are equivalent to the defaults in OMERO 5.0.2 and earlier.

Examples

```
$ omero config set omero.jvmcfg.percent.blitz 50
```

would raise the blitz heap size to 50% of the system memory seen.

```
$ omero config set omero.jvmcfg.system_memory 24000
```

would set the system memory seen to 24GB regardless of the actual amount of memory present in the system. The PercentageStrategy would use this as the basis for setting the Java heap sizes for all services.

```
$ omero config set omero.jvmcfg.max_system_memory 64000
```

would raise the maximum system memory seen by an OMERO installation to 64000MB of system memory. Assuming there was at least 64000MB of memory installed blitz would default to using 9600MB.

```
$ omero config set omero.jvmcfg.strategy.indexer manual
$ omero config set omero.jvmcfg.heap_size.indexer 2000
```

would set the indexer heap size to 2000MB without modifying the settings for the other services.

Tips

View the memory settings that will apply to a newly started server.

```
$ omero admin jvmcfg
```

After modifying any memory settings, be sure to restart your server.

```
$ omero admin restart
```

See also:

<https://www.openmicroscopy.org/community/viewtopic.php?f=4&t=7400>

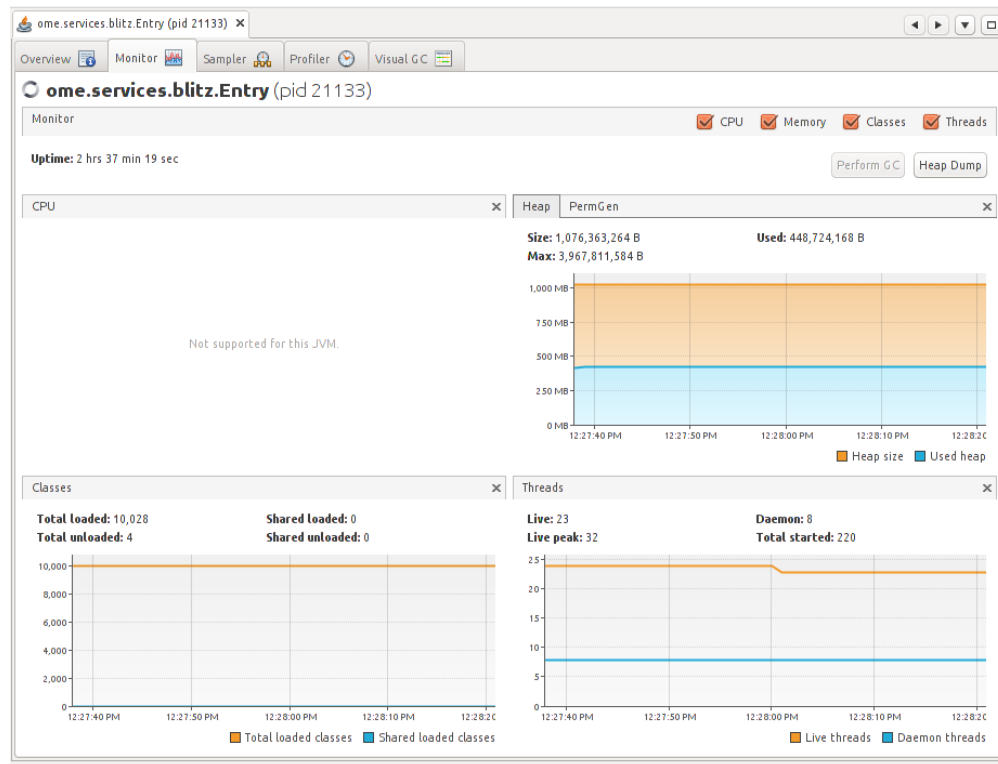
Forum thread on PixelData JVM (Java Virtual Machine) memory settings

Grid configuration

Section of the advanced server configuration documentation describing `etc/grid/templates.xml`.

Monitoring

In addition to watching the OMERO log files, the JVM itself provides a number of tools that you can use to determine the health of your server. [JVisualVM](#), for example, can be used to visualize the memory use of each JVM:



You will need to have the PID (process ID) for the service you want to monitor, which will usually be the main Blitz process. You can find the PID either via `omero admin diagnostics` or alternatively via the `jps` command found in the JDK.

Another tool, [JConsole](#), also provides access to the memory statistics for your JVM, but also lists the JMX (Java Management Extensions) management beans which provide extensive information about the running process. Information includes the number of queries that have been run, the number of open file handles, the system properties that were set on startup, and much more. Further, the [ome.system.metrics](#) package makes use of JMX to expose further properties.

With [further configuration](#), JMX properties can also be accessed remotely which can be very useful for monitoring your server with [Checkmk](#), [Nagios](#), [Zenoss](#), or similar. However, care must be taken to protect the exposed ports.

Note: The commands above require the Java JDK (Java Development Kit) as opposed to the JRE (Java Runtime Environment).

Java Monitoring & Management Console

Connection Window Help

pid: 31778 blitz.jar ome.fulltext --Ice.Config=/opt/ome6/dist/var/master/servers/indexer-0/config/config

Overview Memory Threads Classes VM Summary MBeans

▶ JImplementation
 ▶ bitronix.tm
 ▶ bitronix.tm.resource.jdbc
 ▶ com.sun.management
 ▼ java.lang
 ▶ ClassLoading
 ▶ Compilation
 ▶ GarbageCollector
 ▶ Memory
 ▶ MemoryManager
 ▶ MemoryPool
 ▼ OperatingSystem
 ▶ Attributes
 ▼ Runtime
 ▶ **Attributes**
 ▼ Threading
 ▼ Attributes
 ThreadAllocatedMemoryEnabled
 ThreadAllocatedMemorySupported
 ThreadCount
 TotalStartedThreadCount
 AllThreadIds
 ThreadContentionMonitoringEnabled
 CurrentThreadCpuTime
 CurrentThreadUserTime
 ThreadCpuTimeSupported
 ThreadCpuTimeEnabled
 ThreadContentionMonitoringSupported
 CurrentThreadCpuTimeSupported
 ObjectMonitorUsageSupported
 SynchronizerUsageSupported
 PeakThreadCount
 DaemonThreadCount
 ObjectName
 ▶ Operations
 ▶ java.nio
 ▶ java.util.logging
 ▼ metrics
 ▶ ch.qos.logback.core.Appender.all
 ▼ ch.qos.logback.core.Appender.debug
 ▶ Attributes
 ▶ Operations

Attribute values

Name	Value
BootClassPath	/usr/lib/jvm/java-7-openjdk-amd64/jre/lib...
BootClassPathSupported	true
ClassPath	lib/server/blitz.jar
InputArguments	java.lang.String[6]
LibraryPath	/usr/java/packages/lib/amd64:/usr/lib/x86...
ManagementSpecVersion	1.2
Name	31778@blue
ObjectName	java.lang:type=Runtime
SpecName	Java Virtual Machine Specification
SpecVendor	Oracle Corporation
SpecVersion	1.7
StartTime	1406100482471
SystemProperties	javax.management.openmbean.TabularD...
Uptime	13298106
VmName	OpenJDK 64-Bit Server VM
VmVendor	Oracle Corporation
VmVersion	24.51-b03

Refresh

pid: 31778 blitz.jar ome.fulltext --Ice.Config=/opt/ome6/dist/var/...

Metrics

Building on top of Coda Hale's [Metrics](#) library, OMERO provides the `ome.system.metrics` package which measures a number of internal events and makes them available both via JMX as described under [Monitoring](#) but also prints them to the log files.

By default, these values are printed to each of the JVM-based log files (e.g. `var/log/Blitz-0.log`, `var/log/Indexer-0.log`, etc) once per hour. This value can be configured via `omero.metrics.slf4j_minutes`. A typical value might look like:

```
11:28:18,923 INFO [ome.system.metrics] (r-thread-1) type=TIMER,
name=ome.services.fulltext.FullTextIndexer.batch ...
```

Values include basic statistics (*count*, *min*, *max*, *mean*, etc.) as well as 75th, 90th, 95th, etc percentiles. Further, the rate over the last minute, the last 5 minutes, and the last 15 minutes is provided (*m1*, *m5*, *m15*). For example:

- *count*=3601
- *min*=0.41...
- *max*=7.85...
- *mean*=0.94...
- *stddev*=0.31...
- *median*=0.96...
- *p75*=1.08...
- *p95*=1.25...
- *p98*=1.35...
- *p99*=1.43...
- *p999*=7.69...
- *mean_rate*=0.50...
- *m1*=0.49...
- *m5*=0.499...
- *m15*=0.49...
- *rate_unit*=events/second
- *duration_unit*=milliseconds

Useful metrics include:

ch.qos.logback.core.Appender.error

The number and rate of errors that have been logged. (All services)

jvm.fileDescriptorCountRatio

The ratio of used to available file descriptors. (All services)

ome.services.eventlogs.EventLogQueue.priorityCount

The number of items in the queue. (Indexer-only)

ome.io.nio.PixelsService.minmaxTimes

Time taken to generate min/max values per plane. (PixelData-only)

ome.io.nio.PixelsService.tileTimes

Time taken to generate tiled-pyramids for a big image. (PixelData-only)

2.5.4 Search and indexing configuration

How Indexing works

Indexing is not driven by the user, but happens automatically in the background and can be controlled by a number of settings listed under *Search*. The indexer runs periodically as defined by `omero.search.cron` and parses the latest batch of new or modified objects in the database.

Upon successful completion, the persistent count in the `configuration` table will be incremented.

```
omero=# select value from configuration where name = 'PersistentEventLogLoader.v2.current_
↪id';
 value
-----
 30983
(1 row)
```

Note: Presence of more than one `PersistentEventLogLoader.*` value in your database indicates that you have run indexing with multiple versions of the server. This is fine. To allow a new server version to force an update, the configuration key may be changed. For example, `PersistentEventLogLoader.current_id` became `PersistentEventLogLoader.v2.current_id` in <https://github.com/ome/openmicroscopy/commit/a5cb64a>.

Missing search results

If you are having any difficulty with search results not appearing timely, first you should start by checking the health of the Indexer-0 process:

- Check the server's log directory for a file named `Indexer-0.log` and monitor its progress (e.g. using `tail` or similar). If messages of the format:

```
INFO [ ome.services.fulltext.FullTextIndexer ] (3-thread-2) INDEXED 2 objects in_
↪1 batch(es) [2483 ms.]
```

are periodically being appended to the log file, then your indexer process may be running behind. You can either wait for it to catch up, or try increasing the search batch size in order to speed processing. See the section on the `omero.search.batch` setting for more information.

- If there are no updates to the `Indexer-0.log` file even when new images, tags, files, etc. are added to the server, then it is possible that the Indexer process has become stuck. It is possible to force a restart of the indexer using the *IceGrid Tools* like so:

```
> omero admin ice
Ice 3.6.3 Copyright (c) 2003-2016 ZeroC, Inc.
>>> server list
Blitz-0
DropBox
FileServer
Indexer-0
...
>>> server stop Indexer-0
```

You do not need to manually re-start the Indexer, as IceGrid will handle the creation of a new Indexer process automatically.

In case neither of the above seems to be the case, then your indexer is running normally and more likely your index has been corrupted. You will need to *re-index* OMERO. Reasons why this might have occurred include:

- Missing search terms are part of a very large text file. If the indexer's maximum file size limit is reached, a file may not be indexed. See the section on the `omero.search.max_file_size` setting for more information on increasing this limit.
- A bug in Lucene prior to OMERO 5.0.1 caused some documents to be “sealed” in that old search terms would return the document, but newer terms would **not**.

Re-indexing

Background re-indexing

Under most circumstances, you should be able to re-index the database while the server is still running. If you need to make any adjustments to the server configuration or the process heap size, first shut the server down and make these changes before restarting the server. Use the following steps to initiate a re-indexing.

- Disable the search indexer process and stop any currently running indexer processes:

```
$ omero admin reindex --prepare
```

- Remove the existing search Indexes by deleting the contents of the FullText subdirectory of your `omero.data` dir:

```
$ omero admin reindex --wipe
```

- Reset the indexer's progress counter in the database:

```
$ omero admin reindex --reset 0
```

- Re-enable/restart the indexer process:

```
$ omero admin reindex --finish
```

Depending on the size of your database, it may take the indexer some time to finish re-indexing. During this time, your OMERO server will remain available for use, however the search functionality will be degraded until the re-indexing is finished. See *Monitoring re-indexing* for information on how long this should take.

Note: Once you wipe your full-text directory, searches will return fewer or no results until re-indexing is complete.

Monitoring re-indexing

During re-indexing, it is possible to estimate the percent indexed using the following SQL command:

```
omero=> select 'At ' || current_timestamp(0) || ', Percent indexed: ' || trunc(((select
↳ count(*) from eventlog el, configuration c where el.id < cast(c.value as int) and (c.
↳ name like 'PersistentEventLogLoader%')) * 1.0) / (select count(*) from eventlog) * 100,
↳ 2) || '%';
?column?
```

```
-----
At 2014-06-14 07:54:37+00, Percent indexed: 70.90%
(1 row)
```

This value is also logged periodically when re-indexing in the background and is available via JMX. See [Metrics](#) for more information.

See also:

OMERO search

Section of the developer documentation describing how to perform search queries against the server.

2.5.5 FS configuration options

Background

Users import their image files to the OMERO.fs server. The contents of these files are kept intact by the server and the import process preserves the files' path and name (at least within the rules of `omero.fs.repo.path_rules` below), so that OMERO.fs can become a trusted repository for the master copy of users' data. While the default server configuration from [Configuration properties glossary](#) should typically suffice, **omero config set** may be used to adjust settings related to file uploads. These settings are explained below.

Repository location

Several properties determine where FS-imported files are stored:

- `omero.data.dir` - singleton property (i.e. once globally) which points to the legacy repository location for OMERO. For OMERO to run on multiple systems, the contents of this directory must be on a shared volume.
- `omero.managed.dir` - singleton property which points to the default `ManagedRepository`. In an OMERO install in which there is only one Blitz server, this will be the only repository. This need not be located under `omero.data.dir` but is by default.
- `omero.repo.dir` (experimental) - value passed to all non-legacy, standalone repositories. This is not actively used, but would allow hosting repositories on multiple physical systems without the need for a shared volume. For example, after running `omero admin start` on the main machine, it would be possible to launch nodes on various machines via `omero node start fs-B`, `omero node start fs-C`, etc. Each of these would pass a different `omero.repo.dir` value to its process.

Template path

When files are uploaded to the managed repository, a parent directory is created to receive the upload. A multi-file image has all its files stored in the same parent directory, though they may be in different subdirectories of that parent to mirror the original directory structure before upload. The `omero.fs.repo.path` setting defines the creation of that parent directory. It is this value which makes the `ManagedRepository` “managed”.

Path naming constraints

There is some flexibility in how this parent directory is named. The constraints are:

- The path components (individual directories in the path) must be separated by `/` characters.
- A path component separator may be written as `//` only if followed by at least one more path component. In this case:
 - The server ensures that the path components preceding the `//` are owned by the root user.
 - Any newly created path components following the `//` are **owned by the user** who owns the images.
- If no `//` is present then *all* newly created path components are **owned by the user** who owns the images.

- The path must be unique for each import. It is for this reason that the `%time%` term expands to a time with millisecond resolution.
- To avoid confusion with the expansion terms enumerated below, avoid other uses of the `%` character in path components.

In the above, ownership of path components is in the context of OMERO users accessing the OMERO managed repository through its API. It does not relate to operating system users' permissions for the underlying filesystem.

Expansion terms

Special terms may be used within path components: these are replaced with text that depends on the import.

For any directory in the template path

%userId%

expands to the user's numerical ID

%user%

expands to the user's name

%institution%

expands to the user's institution name; this path component is wholly omitted if the user has no institution set

%institution:default%

expands to the user's institution name, or to the supplied "default" if the user has no institution set; for instance, `%institution:State College of Florida, Manatee-Sarasota%` is permitted

%groupId%

expands to the OMERO group's numerical ID

%group%

expands to the OMERO group's name

%perms%

expands to the group's six-character permissions string, for example `rw----` for a private group

%year%

expands to the current year number, for example `2014`

%month%

expands to the current month number, zero-padded, for example `08`

%monthname%

expands to the current month name, for example `August`

%day%

expands to the current day number in the month, zero-padded, for example `04`

%sessionId%

expands to the session's numerical ID

%session%

expands to the session key (UUID) of the session, for example `6c2dae43-cfad-48ce-af6f-025569f9e6df`

%thread%

expands to the name of the server thread that is performing the import

For user-owned directories only

These expansion terms may not precede `//` in the template path.

%time%

expands to the current time, in hours, minutes, seconds, milliseconds, for example `13-49-07.727`

%hash%

expands to an eight-digit hexadecimal hash code that is constant for the set of files being imported, for example `0554E3A1`

%hash:digits%

expands as `%hash%`, where `digits` is a comma-separated list of how many digits of the hash to use in different subdirectories; for example, `hash-%hash:3,3,2%` expands to a form like `hash-123/456/78`

%increment%

expands to an integer that increases consecutively so as to create the next new directory, for example using `inc-%increment%` with preexisting directories up to `inc-24` would expand to `inc-25`

%increment:digits%

expands as `%increment%` where `digits` specifies a minimum length to which to zero-pad the integer, for example using `inc-%increment:3%` with preexisting directories up to `inc-024` would expand to `inc-025`

%subdirs%

expands to nothing until the preceding directory has more than one thousand entries, in which case it expands to an integer that increases consecutively to similarly limit the entry count in subdirectories; applies recursively to extend the number of path components as needed, so, using `example/below-%subdirs%` in the path, with `example/below-000` to `example/below-999` all “full”, three-digit subdirectories below those are created, such as `example/below-123/456`

%subdirs:digits%

expands as `%subdirs%` where `digits` specifies to how many digits `%subdirs%` may expand for each path component: for example, `example/%subdirs:4%-below` allows ten thousand directory entries in `example` before creating `example/1234-below` and, much later, `example/1234-below/5678`

No more than one of `%time%`, `%subdirs%` or `%increment%` may be used in any one path component, although they may each be used many times in the whole path. If `%subdirs%` expands to nothing then its entire path component is omitted: no other expansion terms in that component are used.

Legal file names

Although OMERO.fs attempts to preserve file naming, the server’s operating system or file system is likely to somehow constrain what file names may be stored by OMERO.fs. This is of particular concern when a user may upload from a more permissive system to a server on a less permissive system, or when it is anticipated that the server itself may be migrated to a less permissive system. The server never accepts Unicode control characters in file names.

The `omero.fs.repo.path_rules` setting defines the combination of restrictions that the server must apply in accepting file uploads. The restrictions are grouped into named sets:

Windows required

prohibits names with the characters `"`, `*`, `/`, `:`, `<`, `>`, `?`, `\`, `|`, names beginning with `$`, the names `AUX`, `CLOCK$`, `CON`, `NUL`, `PRN`, `COM1` to `COM9`, `LPT1` to `LPT9`, and anything beginning with one of those names followed by `.`

Windows optional

prohibits names ending with `.` or a space

UNIX required

prohibits names with the character `/`

UNIX optional

prohibits names beginning with . or -

These rules are applied to each separate path component of the file name on the client's system. So, for instance, an upload of a file `/tmp/myfile.tif` from a Linux system would satisfy the UNIX required restrictions because neither of the path components `tmp` and `myfile.tif` contains a `/` character.

Applying the “optional” restrictions does not assist OMERO.fs at all; those restrictions are designed to ease manual maintenance of the directory specified by the `omero.managed.dir` setting, being where the server stores users' uploaded files.

Checksum algorithm

As the client uploads each file to the server, it calculates a checksum for the file. After the upload is complete the client reports that checksum to the server. The server then calculates the checksum for the corresponding file from its local filesystem and checks that it matches what the client reported. **File integrity** is thus **assured** because corruption during transmission or writing would be revealed by a checksum mismatch.

There are various algorithms by which checksums may be calculated. The list of available algorithms is given by `omero.checksum.supported`. To calculate comparable checksums the client and server use the same algorithm. The server API permits clients to specify the algorithm, but it is expected that they will typically accept the server default.

The number that suffixes each of the checksum algorithm names specifies the bit width of the resulting checksum. A larger bit width makes it less likely that different files will have the same checksum by coincidence, but lengthens the checksum hex strings that are reported to the user and stored in the `hash` column of the `originalfile` table in the database.

2.5.6 Grid configuration

In some cases, the configuration properties will not suffice to fully configure your server. In that case, it may be necessary to make use of IceGrid's XML configuration files. Like the `config.xml` file mentioned above, these are stored under `etc/grid`: `etc/grid/default.xml` is used on Unix systems and there is also `etc/grid/templates.xml`.

The default OMERO application descriptor deploys multiple server instances (Blitz-0, FileServer, Indexer-0, PixelData-0, ...) on a single node. Each server instance is defined by a `server-template` element in `etc/grid/templates.xml` with its own parameters.

Modifying the application descriptors

When you run `omero admin start` without any other arguments, it looks up the default **application descriptor** for your platform:

```
$ omero admin start
No descriptor given. Using etc/grid/default.xml
Waiting on startup. Use CTRL-C to exit
```

The “start” and “deploy” command, however, take several other parameters:

```
$ omero admin start --help
usage: omero admin start [-h] [-u USER] [file] [targets [targets ...]]

Start icegridnode daemon and waits for required components to come up,
```

(continues on next page)

(continued from previous page)

```
i.e. status == 0
```

If the first argument can be found as a file, it will be deployed as the application descriptor rather than `etc/grid/default.xml`. All other arguments will be used as targets to enable optional sections of the descriptor

Positional Arguments:

<code>file</code>	Application descriptor. If not provided, a default will be used
<code>targets</code>	Targets within the application descriptor which should be
<code>↪activated.</code>	

If a file is passed in as the first argument, then that **application descriptor** as opposed to `etc/grid/default.xml` will be used. You can also modify the default application descriptors in place.

Note: The largest issue with using your own application descriptors or modifying the existing ones is that they tend to change between versions, and there is no facility for automatically merging your local changes. You should be prepared to re-make whatever changes you perform directly on the new files.

Targets

Targets are elements within the application descriptors which can optionally turn on configuration. The target is only applicable until the next invocation of **omero admin start** or **omero admin deploy**

Note: You must remember to always apply the targets on each **omero admin** command. If not, the target will not be removed. Therefore, they are often better used for debugging purposes; however, as opposed to alternative application descriptors, using the pre-existing targets should not require any special effort during upgrades.

Debugging

```
<properties id="PythonServer">
  <property name="Ice.ImplicitContext" value="Shared"/>
  <!-- Default logging settings for Python servers. -->
  <property name="omero.logging.timedlog" value="False"/>
  <property name="omero.logging.logsize" value="5000000"/>
  <property name="omero.logging.lognum" value="9"/>
  <property name="omero.logging.level" value="20"/>
  <target name="debug">
    <property name="omero.logging.level" value="10"/>
  </target>
```

Here, the “debug” target allows increasing the logging output of the Python servers without modifying any files.

JMX configuration

```
<server-template id="BlitzTemplate">
  <parameter name="index"/>
  <parameter name="config" default="default"/>
  <parameter name="jmxhost" default=""/>
  <parameter name="jmxport" default="3001"/>
  ...
  <target name="jmx">
    <!-- Be sure to understand the consequences of enabling JMX.
         It allows calling remote methods on your JVM -->
    <option>-Dcom.sun.management.jmxremote=${jmxhost}</option>
    <option>-Dcom.sun.management.jmxremote.port=${jmxport}</option>
    <option>-Dcom.sun.management.jmxremote.authenticate=false</option>
    <option>-Dcom.sun.management.jmxremote.ssl=false</option>
  </target>
```

The JMX target enables remote connections for external monitoring of the Blitz server. If you need to modify the “jmxport” or “jmxhost” variables, you will need to do so directly in the application descriptor XML.

Changing ports / multiple servers on a single host

By modifying the default OMERO ports, it is possible to run multiple OMERO servers on the same physical machine. All port numbers can be adjusted using the relevant *configuration properties*.

To run multiple servers on a single host, the easiest approach is to prefix all ports (SSL, TCP, registry) using `omero.ports.prefix`:

```
# First server
export OMERODIR=~/.OMERO.server-1
omero admin start

# Second server
export OMERODIR=~/.OMERO.server-2
omero config set omero.ports.prefix 1
omero admin start

# Third server
export OMERODIR=~/.OMERO.server-3
omero config set omero.ports.prefix 2
omero admin start
```

Clients will need to use the appropriate port (4064, 14064 or 24064) to connect to OMERO.

See also:

SSL

Section of the *Server security and firewalls* page.

Extending OMERO

Finally, if configuration does not suffice, there are also options for extending OMERO with your own code. These are described on the development site under *Extending OMERO.server*.

2.5.7 Configuration properties glossary

- *Introduction*
- *Mandatory properties*
- *Binary repository*
- *Client*
- *Database*
- *Glacier2*
- *Grid*
- *Ice*
- *JVM*
- *LDAP*
- *Mail*
- *Metrics*
- *Name*
- *Performance*
- *Pixeldata*
- *Policy*
- *Ports*
- *Qa*
- *Query*
- *Scripts*
- *Search*
- *Security*
- *Server*
- *Web*

Introduction

The primary form of configuration is via the use of key/value properties, stored in `etc/grid/config.xml` and read on server startup. Backing up and copying these properties is as easy as copying this file to a new server version.

The <https://github.com/ome/openmicroscopy/blob/develop/etc/omero.properties> file of your distribution defines all the default configuration properties used by the server. Changes made to the file are *not* recognized by the server. Instead, configuration options can be set using the **omero config set** command:

```
$ omero config set <parameter> <value>
```

When supplying a value with spaces or multiple elements, use **single quotes**. The quotes will not be saved as part of the value (see below).

To remove a configuration option (to return to default values where mentioned), simply omit the value:

```
$ omero config set <parameter>
```

These options will be stored in a file: `etc/grid/config.xml` which you can read for reference. **DO NOT** edit this file directly.

You can also review all your settings by using:

```
$ omero config get
```

which should return values without quotation marks.

A final useful option of **omero config edit** is:

```
$ omero config edit
```

which will allow for editing the configuration in a system-default text editor.

Note: Please use the **escape sequence** `\` for nesting double quotes (e.g. `"[\"foo\", \"bar\"]"`) or wrap with `'` (e.g. `'["foo", "bar"]'`).

Examples of doing this are on the [server installation page](#), as well as the [LDAP installation page](#).

Mandatory properties

The following properties need to be correctly set for all installations of the OMERO.server. Depending on your set-up, default values may be sufficient.

- `omero.data.dir`
- `omero.db.host`
- `omero.db.name`
- `omero.db.pass`

Binary repository

property `omero.checksum.supported`

`omero.checksum.supported`

Checksum algorithms supported by the server for new file uploads, being any comma-separated non-empty subset of:

- Adler-32
- CRC-32
- MD5-128
- Murmur3-32
- Murmur3-128
- SHA1-160
- File-Size-64

In negotiation with clients, this list is interpreted as being in descending order of preference.

Default: *SHA1-160, MD5-128, Murmur3-128, Murmur3-32, CRC-32, Adler-32, File-Size-64*

property `omero.data.dir`

`omero.data.dir`

Default: */OMERO/*

property `omero.fs.repo.path`

`omero.fs.repo.path`

Value dynamically set during the build Template for FS managed repository paths. Allowable elements are:

<code>%user%</code>	<code>bob</code>
<code>%userId%</code>	<code>4</code>
<code>%group%</code>	<code>bobLab</code>
<code>%groupId%</code>	<code>3</code>
<code>%year%</code>	<code>2011</code>
<code>%month%</code>	<code>01</code>
<code>%monthname%</code>	<code>January</code>
<code>%day%</code>	<code>01</code>
<code>%time%</code>	<code>15-13-54.014</code>
<code>%institution%</code>	<code>University of Dundee</code>
<code>%hash%</code>	<code>0D2D8DB7</code>
<code>%increment%</code>	<code>14</code>
<code>%subdirs%</code>	<code>023/613</code>
<code>%session%</code>	<code>c3fdd5d8-831a-40ff-80f2-0ba5baef448a</code>
<code>%sessionId%</code>	<code>592</code>
<code>%perms%</code>	<code>rw----</code>
<code>%thread%</code>	<code>Blitz-0-Ice.ThreadPool.Server-3</code>

(continues on next page)

(continued from previous page)

/	path separator
//	end of root-owned directories

These are described further at [FS configuration options](#)

The path must be unique per fileset to prevent upload conflicts, which is why `%time%` includes milliseconds.

A `//` may be used as a path separator: the directories preceding it are created with root ownership, the remainder are the user's. At least one user-owned directory must be included in the path.

The template path is created below `omero.managed.dir`, e.g. `/OMERO/ManagedRepository/$omero.fs.repo.path/`

Default: `%user%_%userId%/%year%-%month%/%day%/%time%`

property `omero.fs.repo.path_rules`

omero.fs.repo.path_rules

Rules to apply to judge the acceptability of FS paths for writing into `omero.managed.dir`, being any comma-separated non-empty subset of:

- Windows required
- Windows optional
- UNIX required
- UNIX optional
- local required
- local optional

Minimally, the “required” appropriate for the server is recommended. Also applying “optional” rules may make sysadmin tasks easier, but may be more burdensome for users who name their files oddly. “local” means “Windows” or “UNIX” depending on the local platform, the latter being applied for Linux and Mac OS X.

Default: *Windows required, UNIX required*

property `omero.managed.dir`

omero.managed.dir

Default: `${omero.data.dir}/ManagedRepository`

Client

property `omero.client.browser.thumb_default_size`

omero.client.browser.thumb_default_size

The default thumbnail size

Default: 96

property omero.client.download_as.max_size

omero.client.download_as.max_size

Clients disable download as jpg/png/tiff above max pixel count.

Default: 144000000

property omero.client.icetransports

omero.client.icetransports

Comma separated list of Ice transports available to clients. The default value ("ssl,tcp") instructs Ice to open the ports specified by the omero.ports.ssl and omero.ports.tcp properties. Restricting to "ssl" will prevent all non-encrypted connections to the OMERO server.

Additionally, there are two experimental values for using websockets: "ws" and "wss" for unencrypted and encrypted, respectively. The ports that are opened are controlled by the omero.ports.ws and omero.ports.wss properties. To enable all possible protocols use: "ssl,tcp,wss,ws".

Note: When using websockets behind a web server like nginx, additional configuration may be needed.

Default: *ssl, tcp*

property omero.client.scripts_to_ignore

omero.client.scripts_to_ignore

Server-side scripts used in IScript service Clients shouldn't display.

Default: */omero/figure_scripts/Movie_Figure.py, /omero/figure_scripts/Split_View_Figure.py,*
/omero/figure_scripts/Thumbnail_Figure.py, /omero/figure_scripts/ROI_Split_Figure.py,
/omero/export_scripts/Make_Movie.py, /omero/import_scripts/Populate_ROI.py

property omero.client.ui.menu.dropdown.colleagues.enabled

omero.client.ui.menu.dropdown.colleagues.enabled

Flag to show/hide colleagues

Default: *true*

property omero.client.ui.menu.dropdown.colleagues.label

omero.client.ui.menu.dropdown.colleagues.label

Client dropdown menu colleagues label.

Default: *Members*

property omero.client.ui.menu.dropdown.everyone.enabled

omero.client.ui.menu.dropdown.everyone.enabled

Flag to show/hide all users.

Default: *true*

property omero.client.ui.menu.dropdown.everyone.label

omero.client.ui.menu.dropdown.everyone.label

Client dropdown menu all users label.

Default: *All Members*

property omero.client.ui.menu.dropdown.leaders.enabled

omero.client.ui.menu.dropdown.leaders.enabled

Flag to show/hide leader.

Default: *true*

property omero.client.ui.menu.dropdown.leaders.label

omero.client.ui.menu.dropdown.leaders.label

Client dropdown menu leader label.

Default: *Owners*

property omero.client.ui.tree.orphans.description

omero.client.ui.tree.orphans.description

Description of the “Orphaned images” container.

Default: *This is a virtual container with orphaned images. These images are not linked anywhere. Just drag them to the selected container.*

property omero.client.ui.tree.orphans.enabled

omero.client.ui.tree.orphans.enabled

Flag to show/hide “Orphaned images” container. Only accept “true” or “false”

Default: *true*

property omero.client.ui.tree.orphans.name

omero.client.ui.tree.orphans.name

Name of the “Orphaned images” container located in client tree data manager.

Default: *Orphaned Images*

property omero.client.ui.tree.type_order

omero.client.ui.tree.type_order

Client tree type order rank list first type is ranked 1 (the highest), last is the lowest if set to ‘false’ empty list allows mixing all types and sorting them by default client ordering strategy

Default: *tagset, tag, project, dataset, screen, plate, acquisition, image*

property omero.client.viewer.initial_zoom_level

omero.client.viewer.initial_zoom_level

Initial client image viewer zoom level for big images

Default: *0*

property omero.client.viewer.interpolate_pixels

omero.client.viewer.interpolate_pixels

Client viewers interpolate pixels by default.

Default: *true*

property omero.client.viewer.roi_limit

omero.client.viewer.roi_limit

Client viewers roi limit.

Default: *2000*

property omero.client.web.host

omero.client.web.host

Absolute omeroweb host `http(s)://your_domain/prefix/`

Default: *[empty]*

Database

property `omero.db.authority`

omero.db.authority

The string that will be used as the base for LSIDs in all exported OME objects including OME-XML and OME-TIFF. It's usually not necessary to modify this value since the database UUID (stored in the database) is sufficient to uniquely identify the source.

Default: *export.openmicroscopy.org*

property `omero.db.dialect`

omero.db.dialect

Implementation of the `org.hibernate.dialect.Dialect` interface which will be used to convert HQL queries and save operations into SQL SELECTs and DML statements.

(PostgreSQL default)

Default: *ome.util.PostgresqlDialect*

property `omero.db.driver`

omero.db.driver

JDBC driver used to access the database. Other drivers can be configured which wrap this driver to provide logging, monitoring, etc.

(PostgreSQL default)

Default: *org.postgresql.Driver*

property `omero.db.host`

omero.db.host

The host name of the machine on which the database server is running. A TCP port must be accessible from the server on which OMERO is running.

Default: *localhost*

property `omero.db.name`

omero.db.name

The name of the database instance to which OMERO will connect.

Default: *omero*

property `omero.db.pass`

omero.db.pass

The password to use to connect to the database server

Default: *omero*

property `omero.db.patch`

omero.db.patch

The patch version of the database which is in use. This value need not match the patch version of the server that is being used with. Any changes by developers to the database schema will result in a bump to this value.

Default: *0*

property `omero.db.poolsize`

omero.db.poolsize

Sets the number of database server connections which will be used by OMERO.

A sizeable increase in this value, e.g. to 100, will significantly increase the performance of your server, but your database installation will need to be configured to accept *at least* as many, preferably more, connections as this value.

The related values `omero.threads.max_threads` and `omero.threads.background_threads` do *not* need to be increased by the same amount. A system will be more stable if `background_threads` is less than `max_threads` and `max_threads` is less than `poolsize`.

Default: *10*

property `omero.db.port`

omero.db.port

TCP port on which the database server is listening for connections. Used by the JDBC driver to access the database. Use of a local UNIX socket is not supported.

(PostgreSQL default)

Default: *5432*

property `omero.db.prepared_statement_cache_size`

omero.db.prepared_statement_cache_size

Default: *10*

property omero.db.profile

omero.db.profile

Default values for the current profile will be hard-coded into the hibernate.properties file in the *model-*.jar*. By using a different jar, you can modify the defaults.

Note: some other properties are defined in the file `etc/profiles/$omero.db.profile` Especially of importance is `omero.db.port` Set during the build

Default: *psql*

property omero.db.properties

omero.db.properties

Properties to set on OMERO.server's JDBC connection to the database. See <https://jdbc.postgresql.org/documentation/use/#connecting-to-the-database>

Default: *[empty]*

property omero.db.sql_action_class

omero.db.sql_action_class

Implementation of the `ome.util.SqlAction` interface which will be used to perform all direct SQL actions, i.e. without Hibernate.

(PostgreSQL default)

Default: *ome.util.actions.PostgresSqlAction*

property omero.db.statistics

omero.db.statistics

Whether JMX statistics are collected for DB usage (by Hibernate, etc)

Default: *true*

property omero.db.url

omero.db.url

The URL specifying how the Java driver connects to the database system.

Default: `jdbc:postgresql://${omero.db.host}:${omero.db.port}/${omero.db.name}?ApplicationName=OMERO.${omero.name}&${ome`

property `omero.db.user`

omero.db.user

The username to use to connect to the database server

Default: `omero`

property `omero.db.version`

omero.db.version

Version of the database which is in use. This value typically matches the major.minor version of the server that it is being used with. Typically, only developers will change this version to bump to a new major version.

Default: `OMERO5.4`

Glacier2

property `omero.glacier2.IceSSL`

omero.glacier2.IceSSL

Glacier2Template IceSSL defaults and overrides, see <https://doc.zeroc.com/ice/3.6/property-reference/icessl>. Any property beginning `omero.glacier2.IceSSL.` will be used to update the corresponding IceSSL. property.

Default: `[empty]`

property `omero.glacier2.IceSSL.Ciphers`

omero.glacier2.IceSSL.Ciphers

Glacier2Template SSL allowed cipher suites

Default: `ADH:!LOW:!MD5:!EXP:!3DES:@STRENGTH`

property `omero.glacier2.IceSSL.ProtocolVersionMax`

omero.glacier2.IceSSL.ProtocolVersionMax

Glacier2Template SSL maximum allowed protocol (mac bug)

Default: *tls1_1*

property omero.glacier2.IceSSL.Protocols

omero.glacier2.IceSSL.Protocols

Glacier2Template SSL allowed protocols

Default: *tls1*

property omero.glacier2.IceSSL.VerifyPeer

omero.glacier2.IceSSL.VerifyPeer

Glacier2Template SSL verification requirements

Default: *0*

Grid

property omero.cluster.read_only

omero.cluster.read_only

Deprecated. If true, will override both the db and repo properties to be true.

Default: *false*

property omero.cluster.read_only.db

omero.cluster.read_only.db

If access to the database is read-only: no writes should be attempted. A “false” may be overridden by omero.cluster.read_only above.

Default: *false*

property omero.cluster.read_only.repo

omero.cluster.read_only.repo

If access to the binary repo is read-only: no writes should be attempted. A “false” may be overridden by `omero.cluster.read_only` above.

Default: *false*

property `omero.cluster.redirector`

omero.cluster.redirector

Default: *nullRedirector*

property `omero.grid.registry_timeout`

omero.grid.registry_timeout

`registry_timeout` is the milliseconds which the registry and other services will wait on remote services to respond.

Default: *5000*

Ice

property `Ice.IPv6`

Ice.IPv6

Disable IPv6 by setting to 0. Only needed in certain situations.

Default: *1*

JVM

property `omero.jvmcfg.append`

omero.jvmcfg.append

Contains other parameters which should be passed to the JVM. The value of “append” is treated as if it were on the command line so will be separated on whitespace. For example, ‘-XX:-PrintGC -XX:+UseCompressedOops’ would result in two new arguments. Note that when using *config set* from the command line one may need to give a prior – option to prevent a value starting with - from already being parsed as an option, and values may need quoting to prevent whitespace or other significant characters from being interpreted prematurely.

Default: *[empty]*

property `omero.jvmcfg.heap_dump`

omero.jvmcfg.heap_dump

Toggles on or off heap dumps on OOMs. Default is “off”. The special value “tmp” will create the heap dumps in your temp directory.

Default: *[empty]*

property omero.jvmcfg.heap_size

omero.jvmcfg.heap_size

Explicit value for the *-Xmx* argument, e.g. “1g”

Default: *[empty]*

property omero.jvmcfg.max_system_memory

omero.jvmcfg.max_system_memory

Suggestion for strategies as to the maximum memory that they will use for calculating JVM settings (MB).

Default: *48000*

property omero.jvmcfg.min_system_memory

omero.jvmcfg.min_system_memory

Suggestion for strategies as to the minimum memory that they will use for calculating JVM settings (MB).

Default: *3414*

property omero.jvmcfg.percent

omero.jvmcfg.percent

Used only by the percent strategy. An integer between 0 and 100 which is the percent of active memory that will be used by the service.

Default: *[empty]*

property omero.jvmcfg.perm_gen

omero.jvmcfg.perm_gen

Explicit value for the MaxPermSize argument to the JVM, e.g. “500M”. Ignored for Java8+

Default: *[empty]*

property omero.jvmcfg.strategy

omero.jvmcfg.strategy

Memory strategy which will be used by default. Options include: percent, manual

Default: *percent*

property `omero.jvmcfg.system_memory`

omero.jvmcfg.system_memory

Manual override of the total system memory that OMERO will *think* is present on the local OS (MB). If unset, an attempt will be made to detect the actual amount: first by using the Python library *psutil* and if that is not installed, by running a Java tool. If neither works, 4.0GB is assumed.

Default: *[empty]*

LDAP

property `omero.ldap.base`

omero.ldap.base

LDAP server base search DN, i.e. the filter that is applied to all users. (can be empty in which case any LDAP user is valid)

Default: *ou=example, o=com*

property `omero.ldap.config`

omero.ldap.config

Enable or disable LDAP (*true* or *false*).

Default: *false*

property `omero.ldap.connect_timeout`

omero.ldap.connect_timeout

Sets `com.sun.jndi.ldap.connect.timeout` on the Spring LDAP default security context source environment. The context source is responsible for interacting with JNDI/LDAP.

This timeout is specified in milliseconds and controls the amount of time JNDI/LDAP will wait for a connection to be established.

A timeout less than or equal to zero means that no timeout will be observed and that the OMERO server will wait indefinitely for LDAP connections to be established. Such a timeout should be used with extreme caution as connectivity issues may then cause your server to no longer be able to create new sessions.

For more information on what this JNDI/LDAP property does, see <https://docs.oracle.com/javase/jndi/tutorial/ldap/connect/create.html>

Default: *5000*

property omero.ldap.group_filter

omero.ldap.group_filter

Default: (*objectClass=groupOfNames*)

property omero.ldap.group_mapping

omero.ldap.group_mapping

Default: *name=cn*

property omero.ldap.new_user_group

omero.ldap.new_user_group

Without a prefix the “new_user_group” property specifies the name of a single group which all new users will be added to. Other new_user_group strings are prefixed with `:x:` and specify various lookups which should take place to find one or more target groups for the new user.

`:ou:` uses the final organizational unit of a user’s dn as the single OMERO group e.g. `omero.ldap.new_user_group=:ou:`

`:attribute:` uses all the values of the specified attribute as the name of multiple OMERO groups. e.g. `omero.ldap.new_user_group=:attribute:memberOf`

Like `:attribute:`, `:filtered_attribute:` uses all the values of the specified attribute as the name of multiple OMERO groups but the attribute must pass the same filter as `:query:` does. e.g. `omero.ldap.new_user_group=:filtered_attribute:memberOf`

Similar to `:attribute:`, `:dn_attribute:` uses all the values of the specified attribute as the DN of multiple OMERO groups. e.g. `omero.ldap.new_user_group=:dn_attribute:memberOf`

A combination of `filtered_attribute` and `dn_attribute`, `:filtered_dn_attribute:` uses all of the values of the specified attribute as the DN of multiple OMERO groups but the attribute must pass the same filter as `:query:` e.g. `omero.ldap.new_user_group=:filtered_dn_attribute:memberOf`

`:query:` performs a query for groups. The “name” property will be taken as defined by `omero.ldap.group_mapping` and the resulting filter will be AND’ed with the value `group_filter` (above) e.g. `omero.ldap.new_user_group=:query:(member=@{dn})`

`:bean:` looks in the server’s context for a bean with the given name which implements `ome.security.auth.NewUserGroupBean` e.g. `omero.ldap.new_user_group=:bean:myNewUserGroupMapperBean`

Default: *default*

property omero.ldap.new_user_group_owner

omero.ldap.new_user_group_owner

A query element to check if user who is being created via the `new_user_group` setting should be made a “manager”, i.e. owner, of the queried group. E.g. `omero.ldap.new_user_group_owner=(owner=@{dn})` will use the ‘manager’ attribute to set the ‘owner’ flag in the database. This query element is appended to any query used by `new_user_group` with an AND.

This property is not used by `new_user_group` type ‘default’ and only potentially by `:bean:`.

Default: *[empty]*

property `omero.ldap.password`

omero.ldap.password

LDAP server bind password (if required; can be empty)

Default: *[empty]*

property `omero.ldap.read_timeout`

omero.ldap.read_timeout

Sets `com.sun.jndi.ldap.read.timeout` on the Spring LDAP default security context source environment. The context source is responsible for interacting with JNDI/LDAP.

This timeout is specified in milliseconds and controls the amount of time JNDI/LDAP will wait for a response from the LDAP server. When connecting to a server using SSL this timeout also applies to the SSL handshake process.

A timeout less than or equal to zero means that no timeout will be observed and that the OMERO server will wait indefinitely for LDAP replies. Such a timeout should be used with extreme caution, especially when using SSL and/or without a connection pool, as connectivity issues may then cause your server to no longer be able to create new sessions.

For more information on what this JNDI/LDAP property does, see <https://docs.oracle.com/javase/tutorial/jndi/newstuff/readtimeout.html>

Default: *5000*

property `omero.ldap.referral`

omero.ldap.referral

Available referral options are: “ignore”, “follow”, or “throw” as per the JNDI referral documentation.

Default: *ignore*

property `omero.ldap.sync_on_login`

omero.ldap.sync_on_login

Whether or not values from LDAP will be synchronized to OMERO on each login. This includes not just the username, email, etc, but also the groups that the user is a member of.

Note: Admin actions carried out in the clients may not survive this synchronization e.g. LDAP users removed from an LDAP group in the UI will be re-added to the group when logging in again after the synchronization.

Default: *false*

property omero.ldap.urls

omero.ldap.urls

Set the URL of the LDAP server. A SSL URL for this property would be of the form: ldaps://ldap.example.com:636

Default: *ldap://localhost:389*

property omero.ldap.user_filter

omero.ldap.user_filter

Default: *(objectClass=person)*

property omero.ldap.user_mapping

omero.ldap.user_mapping

Default: *omeName=cn, firstName=givenName, lastName=sn, email=mail, institution=department, middleName=middleName*

property omero.ldap.username

omero.ldap.username

LDAP server bind DN (if required; can be empty)

Default: *[empty]*

Mail

property omero.mail.bean

omero.mail.bean

Mail sender properties

Default: *defaultMailSender*

property omero.mail.config

omero.mail.config

Enable or disable mail sender (*true* or *false*).

Default: *false*

property omero.mail.from

omero.mail.from

the email address used for the “from” field

Default: *omero@\${omero.mail.host}*

property omero.mail.host

omero.mail.host

the hostname of smtp server

Default: *localhost*

property omero.mail.password

omero.mail.password

the password to connect to the smtp server (if required; can be empty)

Default: *[empty]*

property omero.mail.port

omero.mail.port

the port of smtp server

Default: 25

property omero.mail.smtp.auth

omero.mail.smtp.auth

see javax.mail.Session properties

Default: *false*

property omero.mail.smtp.connectiontimeout

omero.mail.smtp.connectiontimeout

Default: *60000*

property omero.mail.smtp.debug

omero.mail.smtp.debug

Default: *false*

property omero.mail.smtp.socketFactory.class

omero.mail.smtp.socketFactory.class

Default: *javax.net.SocketFactory*

property omero.mail.smtp.socketFactory.fallback

omero.mail.smtp.socketFactory.fallback

Default: *false*

property omero.mail.smtp.socketFactory.port

omero.mail.smtp.socketFactory.port

Default: *\${omero.mail.port}*

property omero.mail.smtp.starttls.enable

omero.mail.smtp.starttls.enable

Default: *false*

property omero.mail.smtp.timeout

omero.mail.smtp.timeout

Default: *60000*

property `omero.mail.transport.protocol`

omero.mail.transport.protocol

other smtp parameters; see `org.springframework.mail.javamail.JavaMailSenderImpl`

Default: *smtp*

property `omero.mail.username`

omero.mail.username

the username to connect to the smtp server (if required; can be empty)

Default: *[empty]*

Metrics

property `omero.metrics.bean`

omero.metrics.bean

Which bean to use: `nullMetrics` does nothing `defaultMetrics` uses the properties defined below

Default: *defaultMetrics*

property `omero.metrics.graphite`

omero.metrics.graphite

Address for Metrics to send server data

Default: *[empty]*

property `omero.metrics.slf4j_minutes`

omero.metrics.slf4j_minutes

Number of minutes to periodically print to slf4j 0 or lower disables the printout.

Default: *60*

Name

property `omero.name`

`omero.name`

Name of the OMERO component that is running in this process.

Default: *Server*

Performance

property `omero.sessions.max_user_time_to_idle`

`omero.sessions.max_user_time_to_idle`

Sets the maximum duration in milliseconds a user can request before a login is required due to inactivity.

Default: *6000000*

property `omero.sessions.max_user_time_to_live`

`omero.sessions.max_user_time_to_live`

Sets the maximum duration in milliseconds a user can request before a login is required (0 signifies never).

Default: *0*

property `omero.sessions.maximum`

`omero.sessions.maximum`

Sets the default duration before a login is required; 0 signifies never.

Default: *0*

property `omero.sessions.sync_force`

`omero.sessions.sync_force`

Default: *1800000*

property `omero.sessions.sync_interval`

omero.sessions.sync_interval

Default: *120000*

property `omero.sessions.timeout`

omero.sessions.timeout

Sets the default duration of inactivity in milliseconds after which a login is required.

Default: *600000*

property `omero.threads.background_threads`

omero.threads.background_threads

Number of threads from the `min_threads` pool that can be used at any given time for background tasks like import. Note that if this value is less than `min_threads`, `min_threads` will limit the number of background tasks which can run simultaneously.

Default: *10*

property `omero.threads.background_timeout`

omero.threads.background_timeout

Number of milliseconds to wait for a slot in the background queue before a rejection error will be raised.

Default: *3600000*

property `omero.threads.cancel_timeout`

omero.threads.cancel_timeout

Default: *5000*

property `omero.threads.idle_timeout`

omero.threads.idle_timeout

This setting does nothing. See <https://github.com/ome/omero-server/issues/154> And <https://github.com/ome/omero-server/pull/155>

Default: *5000*

property `omero.threads.max_threads`

omero.threads.max_threads

This setting does nothing. See <https://github.com/ome/omero-server/issues/154> And <https://github.com/ome/omero-server/pull/155>

Default: 50

property `omero.threads.min_threads`

omero.threads.min_threads

Maximum and minimum number of threads that can simultaneously run at the “USER” and “BACKGROUND” priority level. Internal system threads may still run. Note when setting this that these threads do not time out.

Default: 5

property `omero.throttling.method_time.error`

omero.throttling.method_time.error

Time in milliseconds after which a single method invocation will print a ERROR statement to the server log. If ERRORs are frequently being printed to your logs, you may want to increase this value after checking that no actual problem exists. Values of more than 60000 (1 minute) are not advised.

Default: 20000

property `omero.throttling.method_time.error.indexer`

omero.throttling.method_time.error.indexer

Value for the indexer is extended to 1 day

Default: 86400000

property `omero.throttling.method_time.warn`

omero.throttling.method_time.warn

Time in milliseconds after which a single method invocation will print a WARN statement to the server log.

Default: 5000

property `omero.throttling.method_time.warn.indexer`

omero.throttling.method_time.warn.indexer

Value for the indexer is extended to 1 hour

Default: *3600000*

property omero.throttling.objects_read_interval

omero.throttling.objects_read_interval

Default: *1000*

property omero.throttling.objects_written_interval

omero.throttling.objects_written_interval

Default: *1000*

property omero.throttling.servants_per_session

omero.throttling.servants_per_session

Default: *10000*

Pixeldata

property omero.pixeldata.backoff

omero.pixeldata.backoff

Name of the spring bean which will be used to calculate the backoff (in ms) that users should wait for an image to be ready to view.

Default: *ome.io.nio.SimpleBackOff*

property omero.pixeldata.backoff.default

omero.pixeldata.backoff.default

A default value for the backoff time.

Default: *1000*

property omero.pixeldata.backoff.maxpixels

omero.pixeldata.backoff.maxpixels

The maximum number of pixels (in any dimension), if exceeded the default value will be used.

Default: *1000000*

property omero.pixeldata.batch

omero.pixeldata.batch

Number of instances indexed per indexing. (Ignored by pixelDataEventLogQueue)

Default: *50*

property omero.pixeldata.cron

omero.pixeldata.cron

Polling frequency of the pixeldata processing. Set empty to disable pixeldata processing.

Cron Format: seconds minutes hours day-of-month month day-of-week year (optional). For example, “0,30 * * * * ?” is equivalent to running every 30 seconds. See <https://www.quartz-scheduler.org/api/1.8.6/org/quartz/CronExpression.html>

Default: **/4 * * * * ?*

property omero.pixeldata.dispose

omero.pixeldata.dispose

Whether the PixelData.dispose() method should try to clean up ByteBuffer instances which may lead to memory exceptions. See ticket #11675 for more information. Note: the property is set globally for the JVM.

Default: *true*

property omero.pixeldata.event_log_loader

omero.pixeldata.event_log_loader

EventLogLoader that will be used for loading EventLogs for the action “PIXELDATA”. Choices include: pixelDataEventLogQueue and the older pixelDataPersistentEventLogLoader

Default: *pixelDataEventLogQueue*

property omero.pixeldata.max_plane_float_override

omero.pixeldata.max_plane_float_override

If true, server will not require pyramids for floating point pixel types. Note that pyramids are never generated for floating point pixel types.

Default: *true*

property omero.pixeldata.max_plane_height

omero.pixeldata.max_plane_height

With omero.pixeldata.max_plane_width, specifies the plane size cutoff above which a pixel pyramid will be generated by the pixeldata service unless subresolutions can be read from the file format. These values will be ignored for floating or double pixel data types unless :property: *omero.pixeldata.max_plane_float_override* is set to false. Note pyramids are never generated for floating point pixel types.

Default: *3192*

property omero.pixeldata.max_plane_width

omero.pixeldata.max_plane_width

With omero.pixeldata.max_plane_height, specifies the plane size cutoff above which a pixel pyramid will be generated by the pixeldata service unless subresolutions can be read from the file format. These values will be ignored for floating or double pixel data types unless :property: *omero.pixeldata.max_plane_float_override* is set to false. Note pyramids are never generated for floating point pixel types.

Default: *3192*

property omero.pixeldata.max_projection_bytes

omero.pixeldata.max_projection_bytes

Specifies the maximum number of bytes the server will allow to be projected in real time with the rendering engine.

Default: *268435456*

property omero.pixeldata.memoizer.dir

omero.pixeldata.memoizer.dir

The directory in which Bio-Formats may create memo files for images from the managed repository.

Default: *\${omero.data.dir}/BioFormatsCache*

property omero.pixeldata.memoizer.dir.local

omero.pixeldata.memoizer.dir.local

For read-only servers set this to a local read-write directory so that memo files can be created and used. Activates only if the binary repository is read-only.

Default: *[empty]*

property omero.pixeldata.memoizer_wait

omero.pixeldata.memoizer_wait

Maximum time in milliseconds that file parsing can take without the parsed metadata being cached to omero.pixeldata.memoizer.dir.

Default: *0*

property omero.pixeldata.repetitions

omero.pixeldata.repetitions

Instead, it is possible to tell the server to run more pixeldata repetitions, each of which gets completely committed before the next. This will only occur when there is a substantial backlog of pixels to process.

(Ignored by pixelDataEventLogQueue; uses threads instead)

Default: *1*

property omero.pixeldata.threads

omero.pixeldata.threads

How many pixel pyramids will be generated at a single time. The value should typically not be set to higher than the number of cores on the server machine.

Default: *2*

property omero.pixeldata.tile_height

omero.pixeldata.tile_height

Default: *256*

property omero.pixeldata.tile_sizes_bean

omero.pixeldata.tile_sizes_bean

Default sizes for tiles are provided by a `ome.io.nio.TileSizes` implementation. By default the bean (“configuredTileSizes”) uses the properties provided here.

Default: *configuredTileSizes*

property `omero.pixeldata.tile_width`

omero.pixeldata.tile_width

Default: 256

Policy

property `omero.policy.bean`

omero.policy.bean

Instance of the `PolicyService` interface which will be responsible for checking certain server actions made by a user.

Default: *defaultPolicyService*

property `omero.policy.binary_access`

omero.policy.binary_access

Configuration for the policy of whether users can access binary files from disk. Binary access includes all attempts to download a file from the UI.

The individual components of the string include:

- write - whether or not users who have WRITE access to the objects can access the binary. This includes group and system administrators.
- read - whether or not users who have READ access to the objects can access the binary.
- image - whether or not images are to be considered accessible as a rule.
- plate - whether or not plates and contained HCS objects are to be considered accessible as a rule. This includes wells, well samples, and plate runs.

Though the order of the components of the property are not important, the order that they are listed above roughly corresponds to their priority. E.g. a -write value will override +plate.

Example 1: “-read,+write,+image,-plate” only owners of an image and admins can download it.

Example 2: “-read,-write,-image,-plate” no downloading is possible.

Configuration properties of the same name can be applied to individual groups as well. E.g. adding, `omero.policy.binary_access=-read` to a group’s `config` property, you can prevent group-members from downloading original files, as at <https://docs.openmicroscopy.org/latest/omero/sysadmins/customization.html#download-restrictions>

Configuration is pessimistic: if there is a negative *either* on the group *or* at the server-level, the restriction will be applied. A missing value at the server restricts the setting but allows the server to override.

Default: *+read, +write, +image*

Ports

property `omero.ports.prefix`

`omero.ports.prefix`

The prefix to apply to all port numbers (SSL, TCP, registry) used by the server

Default: *[empty]*

property `omero.ports.registry`

`omero.ports.registry`

The IceGrid registry port number to use

Default: *4061*

property `omero.ports.ssl`

`omero.ports.ssl`

The Glacier2 SSL port number to use

Default: *4064*

property `omero.ports.tcp`

`omero.ports.tcp`

The Glacier2 TCP port number to use (unencrypted)

Default: *4063*

property `omero.ports.ws`

`omero.ports.ws`

The Glacier2 WS port number to use (unencrypted)

Default: *4065*

property `omero.ports.wss`

omero.ports.wss

The Glacier2 WSS port number to use

Default: *4066*

Qa

property `omero.qa.feedback`

omero.qa.feedback

Base URL to use when sending feedback (errors, comments)

Default: *http://qa.openmicroscopy.org.uk*

Query

property `omero.query.timeout`

omero.query.timeout

For the query service how many seconds before a query times out.

Default: *1000*

property `omero.query.timeout.admin`

omero.query.timeout.admin

How many seconds before a query times out for administrative users.

Default: *\${omero.query.timeout}*

Scripts

property `omero.launcher.jython`

omero.launcher.jython

Executable on the PATH which will be used for scripts with the mimetype 'text/x-jython'.

Default: *jython*

property `omero.launcher.matlab`

omero.launcher.matlab

Executable on the PATH which will be used for scripts with the mimetype 'text/x-matlab'.

Default: *matlab*

property omero.launcher.python

omero.launcher.python

Executable on the PATH which will be used for scripts with the mimetype 'text/x-python'.

No value implies use sys.executable

Default: *[empty]*

property omero.process.jython

omero.process.jython

Server implementation which will be used for scripts with the mimetype 'text/x-jython'. Changing this value requires that the appropriate class has been installed on the server.

Default: *omero.processor.ProcessI*

property omero.process.matlab

omero.process.matlab

Server implementation which will be used for scripts with the mimetype 'text/x-matlab'. Changing this value requires that the appropriate class has been installed on the server.

Default: *omero.processor.MATLABProcessI*

property omero.process.python

omero.process.python

Server implementation which will be used for scripts with the mimetype 'text/x-python'. Changing this value requires that the appropriate class has been installed on the server.

Default: *omero.processor.ProcessI*

property omero.scripts.cache.cron

omero.scripts.cache.cron

Frequency to reload script params. By default, once a day at midnight.

Cron Format: seconds minutes hours day-of-month month day-of-week year (optional). For example, “0,30 * * * * ?” is equivalent to running every 30 seconds. See <https://www.quartz-scheduler.org/api/1.8.6/org/quartz/CronExpression.html>

Default: *0 0 0 * * ?*

property `omero.scripts.cache.spec`

omero.scripts.cache.spec

Guava LoadingCache spec for configuring how many script JobParams will be kept in memory for how long.

For more information, see <https://google.github.io/guava/releases/27.1-jre/api/docs/com/google/common/cache/CacheBuilderSpec.html>

Default: *maximumSize=1000*

property `omero.scripts.timeout`

omero.scripts.timeout

Default: *3600000*

Search

property `omero.search.analyzer`

omero.search.analyzer

Analyzer used both index and to parse queries

Default: *ome.services.fulltext.FullTextAnalyzer*

property `omero.search.batch`

omero.search.batch

Size of the batches to process events per indexing. Larger batches can speed up indexing, but at the cost of memory.

Default: *5000*

property `omero.search.bridges`

omero.search.bridges

Extra bridge classes, comma-separated, to be invoked on each indexing. Bridges are used to parse more information out of the data.

Default: *[empty]*

property omero.search.cron

omero.search.cron

Polling frequency of the indexing. Set empty to disable search indexing.

Cron Format: seconds minutes hours day-of-month month day-of-week year (optional). For example, “0,30 * * * * ?” is equivalent to running every 30 seconds. See <https://www.quartz-scheduler.org/api/1.8.6/org/quartz/CronExpression.html>

Default: **/2 * * * * ?*

property omero.search.event_log_loader

omero.search.event_log_loader

Default: *eventLogQueue*

property omero.search.excludes

omero.search.excludes

Indexing takes place on all EventLogs as they occur in the database. The types listed here will be skipped if they appear in the “entityType” field of the EventLog table.

Default: *ome.model.annotations.ChannelAnnotationLink, ome.model.core.Channel, ome.model.core.PlaneInfo, ome.model.core.PixelsOriginalFileMap, ome.model.containers.DatasetImageLink, ome.model.containers.ProjectDatasetLink, ome.model.containers.CategoryGroupCategoryLink, ome.model.containers.CategoryImageLink, ome.model.display.ChannelBinding, ome.model.display.QuantumDef, ome.model.display.Thumbnail, ome.model.meta.Share, ome.model.meta.Event, ome.model.meta.EventLog, ome.model.meta.GroupExperimenterMap, ome.model.meta.Node, ome.model.meta.Session, ome.model.annotations.RoiAnnotationLink, ome.model.roi.Roi, ome.model.roi.Shape, ome.model.roi.Text, ome.model.roi.Rectangle, ome.model.roi.Mask, ome.model.roi.Ellipse, ome.model.roi.Point, ome.model.roi.Path, ome.model.roi.Polygon, ome.model.roi.Polyline, ome.model.roi.Line, ome.model.screen.ScreenAcquisitionWellSampleLink, ome.model.screen.ScreenPlateLink, ome.model.screen.WellReagentLink, ome.model.stats.StatsInfo*

property omero.search.include_actions

omero.search.include_actions

EventLog.action values which will be indexed. Unless custom code is generating other action types, this property should not need to be modified.

Default: *INSERT, UPDATE, REINDEX, DELETE*

property omero.search.include_types

omero.search.include_types

Whitelist of object types which will be indexed. All other types will be ignored. This matches the currently available UI options but may need to be expanded for custom search bridges.

Default: *ome.model.core.Image, ome.model.containers.Project, ome.model.containers.Dataset, ome.model.screen.Plate, ome.model.screen.Screen, ome.model.screen.PlateAcquisition, ome.model.screen.Well*

property omero.search.locking_strategy

omero.search.locking_strategy

Default: *native*

property omero.search.max_file_size

omero.search.max_file_size

Maximum file size for text indexing (bytes) If a file larger than this is attached, e.g. to an image, the indexer will simply ignore the contents of the file when creating the search index. This should not be set to more than half of the Indexer heap space.

Note: If you set the max file size to greater than 1/2 the size of the indexer's heap (256 MB by default), you may encounter Out of Memory errors in the Indexer process or you may cause the search index to become corrupt. Be sure that you also increase the heap size accordingly (see *OutOfMemoryError / PermGen space errors in OMERO.server logs*).

Default: *131072000*

property omero.search.max_fileset_size

omero.search.max_fileset_size

Maximum number of fileset entries which will be indexed Increasing this cut-off can lead to indexing performance degradation notably in the high-content screening domain where plates typically contain 1K-10K images associated with 10-100K fileset entries each If set to 0, no fileset entry will be indexed

Default: *10*

property omero.search.max_partition_size

omero.search.max_partition_size

Number of objects to load in a single indexing window. The larger this value the fewer times a single object will be indexed unnecessarily. Each object uses roughly 100 bytes of memory.

Default: *1000000*

property omero.search.merge_factor

omero.search.merge_factor

Default: *25*

property omero.search.ram_buffer_size

omero.search.ram_buffer_size

Default: *64*

property omero.search.repetitions

omero.search.repetitions

Instead, it is possible to tell the server to run more indexing repetitions, each of which gets completely committed before the next. This will only occur when there is a substantial backlog of searches to perform. (More than 1 hours worth)

Default: *1*

property omero.search.reporting_loops

omero.search.reporting_loops

Periodically the completion percentage will be printed. The calculation can be expensive and so is not done frequently.

Default: *100*

Security

property omero.security.chmod_strategy

omero.security.chmod_strategy

Default: *groupChmodStrategy*

property omero.security.filter.bitand

omero.security.filter.bitand

Default: *(int&and(permissions, %s) = %s)*

property omero.security.keyStore

omero.security.keyStore

A keystore is a database of private keys and their associated X.509 certificate chains authenticating the corresponding public keys. A keystore is mostly needed if you are doing client-side certificates for authentication against your LDAP server.

Default: *[empty]*

property omero.security.keyStorePassword

omero.security.keyStorePassword

Sets the password of the keystore

Default: *[empty]*

property omero.security.login_failure_throttle_count

omero.security.login_failure_throttle_count

Default: *1*

property omero.security.login_failure_throttle_time

omero.security.login_failure_throttle_time

Default: *3000*

property omero.security.password_provider

omero.security.password_provider

Implementation of PasswordProvider that will be used to authenticate users. Typically, a chained password provider will be used so that if one form of authentication (e.g. LDAP) does not work, other attempts will be made.

Default: *chainedPasswordProvider*

property omero.security.password_required

omero.security.password_required

Controls whether the server will allow creation of user accounts with an empty password. If set to true (default, strict mode), empty passwords are disallowed. This still allows the guest user to interact with the server.

Default: *true*

property omero.security.trustStore

omero.security.trustStore

A truststore is a database of trusted entities and their associated X.509 certificate chains authenticating the corresponding public keys. The truststore contains the Certificate Authority (CA) certificates and the certificate(s) of the other party to which this entity intends to send encrypted (confidential) data. This file must contain the public key certificates of the CA and the client's public key certificate.

Default: *[empty]*

property omero.security.trustStorePassword

omero.security.trustStorePassword

Sets the password of the truststore

Default: *[empty]*

Server

property omero.server.nodedescriptors

omero.server.nodedescriptors

Override the default set of OMERO services. For example, to run OMERO.server with Blitz and Tables only (i.e. disable Processor, DropBox, Indexer, PixelData) set this to `master:Blitz-0,Tables-0`. Also use this to distribute OMERO services across multiple nodes, for example: `master:Blitz-0,Tables-0 worker1:Processor-0`. See <https://docs.openmicroscopy.org/omero/latest/sysadmins/grid.html#deployment-examples>

Default: *[empty]*

Web

property omero.web.admins

omero.web.admins

A list of people who get code error notifications whenever the application identifies a broken link or raises an unhandled exception that results in an internal server error. This gives the administrators immediate notification of any errors, see [OMERO.mail](#). Example: '[["Full Name", "email address"]]'.

Default: *[]*

property omero.web.application_server

omero.web.application_server

OMERO.web is configured to run in Gunicorn as a generic WSGI (TCP) application by default. Available options: `wsgi-tcp` (Gunicorn, default), `wsgi` (Advanced users only, e.g. manual Apache configuration with `mod_wsgi`).

Default: *wsgi-tcp*

property omero.web.application_server.host

omero.web.application_server.host

The front-end webserver e.g. NGINX can be set up to run on a different host from OMERO.web. The property ensures that OMERO.web is accessible on an external IP. It requires copying all the OMERO.web static files to the separate NGINX server.

Default: *127.0.0.1*

property omero.web.application_server.max_requests

omero.web.application_server.max_requests

The maximum number of requests a worker will process before restarting.

Default: *0*

property omero.web.application_server.port

omero.web.application_server.port

Upstream application port

Default: *4080*

property omero.web.apps

omero.web.apps

Add additional Django applications. For example, see *Creating an app*

Default: `[]`

property `omero.web.base_include_template`

omero.web.base_include_template

Template to be included in every page, at the end of the `<body>`

Default: *None*

property `omero.web.caches`

omero.web.caches

OMERO.web offers alternative session backends to automatically delete stale data using the cache session store backend, see [Django cached session documentation](#) for more details.

Default: `{ "default": { "BACKEND": "django.core.cache.backends.dummy.DummyCache" }}`

property `omero.web.chunk_size`

omero.web.chunk_size

Size, in bytes, of the “chunk”

Default: `1048576`

property `omero.web.cors_origin_allow_all`

omero.web.cors_origin_allow_all

If True, `cors_origin_whitelist` will not be used and all origins will be authorized to make cross-site HTTP requests.

Default: *false*

property `omero.web.cors_origin_whitelist`

omero.web.cors_origin_whitelist

A list of origin hostnames that are authorized to make cross-site HTTP requests. Used by the `django-cors-headers` app as described at <https://github.com/ottoyiu/django-cors-headers>

Default: `[]`

property `omero.web.csrf_cookie_httponly`

omero.web.csrf_cookie_httponly

Prevent CSRF cookie from being accessed in JavaScript. Currently disabled as it breaks background JavaScript POSTs in OMERO.web.

Default: *false*

property `omero.web.csrf_cookie_samesite`

omero.web.csrf_cookie_samesite

The value of the SameSite flag on the CSRF cookie. This flag prevents the cookie from being sent in cross-site requests thus preventing CSRF attacks and making some methods of CSRF session cookie impossible.

Default: *Lax*

property `omero.web.csrf_cookie_secure`

omero.web.csrf_cookie_secure

Restrict CSRF cookies to HTTPS only, you are strongly recommended to set this to `true` in production.

Default: *false*

property `omero.web.csrf_trusted_origins`

omero.web.csrf_trusted_origins

A list of hosts which are trusted origins for unsafe requests. When starting with '.', all subdomains are included. Example `['.example.com', 'another.example.net']`. For more details see [CSRF trusted origins](#).

Default: *[]*

property `omero.web.databases`

omero.web.databases

Default: *{}*

property `omero.web.debug`

omero.web.debug

A boolean that turns on/off debug mode. Use debug mode only in development, not in production, as it logs sensitive and confidential information in plaintext.

Default: *false*

property `omero.web.django_additional_settings`

omero.web.django_additional_settings

Additional Django settings as list of key-value tuples. Use this to set or override Django settings that aren't managed by OMERO.web. E.g. ["CUSTOM_KEY", "CUSTOM_VALUE"]

Default: []

property omero.web.favicon_url

omero.web.favicon_url

Favicon URL, specifies the path relative to django's static file dirs.

Default: *webgateway/img/ome.ico*

property omero.web.feedback.comment.enabled

omero.web.feedback.comment.enabled

Enable the feedback form for comments. These comments are sent to the URL in omero.qa.feedback (OME team by default).

Default: *true*

property omero.web.feedback.error.enabled

omero.web.feedback.error.enabled

Enable the feedback form for errors. These errors are sent to the URL in omero.qa.feedback (OME team by default).

Default: *true*

property omero.web.html_meta_referrer

omero.web.html_meta_referrer

Default content for the HTML Meta referrer tag. See <https://www.w3.org/TR/referrer-policy/#referrer-policies> for allowed values and <https://caniuse.com/referrer-policy> for browser compatibility. Warning: Internet Explorer 11 does not support the default value of this setting, you may want to change this to "origin" after reviewing the linked documentation.

Default: *origin-when-crossorigin*

property omero.web.index_template

omero.web.index_template

Define template used as an index page `http://your_host/omero/`. If None user is automatically redirected to the login page. For example use `'webclient/index.html'`.

Default: *None*

property `omero.web.logdir`

omero.web.logdir

A path to the custom log directory.

Default: */home/omero/OMERO.server/var/log*

property `omero.web.login.client_downloads_base`

omero.web.login.client_downloads_base

GitHub repository containing the Desktop client downloads

Default: *ome/omero-insight*

property `omero.web.login.show_client_downloads`

omero.web.login.show_client_downloads

Whether to link to official client downloads on the login page

Default: *true*

property `omero.web.login_incorrect_credentials_text`

omero.web.login_incorrect_credentials_text

The error message shown to users who enter an incorrect username or password.

Default: *Connection not available, please check your user name and password.*

property `omero.web.login_logo`

omero.web.login_logo

Customize webclient login page with your own logo. Logo images should ideally be 150 pixels high or less and will appear above the OMERO logo. You will need to host the image somewhere else and link to it with the OMERO logo.

Default: *None*

property `omero.web.login_redirect`

omero.web.login_redirect

Redirect to the given location after logging in. It only supports arguments for [Django reverse function](#). For example: `'{"redirect": ["webindex"], "viewname": "load_template", "args":["userdata"], "query_string": {"experimenter": -1}}'`

Default: `{}`

property omero.web.login_view

omero.web.login_view

The Django view name used for login. Use this to provide an alternative login workflow.

Default: `weblogin`

property omero.web.max_table_download_rows

omero.web.max_table_download_rows

Prevent download of OMERO.tables exceeding this number of rows in a single request.

Default: `10000`

property omero.web.maximum_multifile_download_size

omero.web.maximum_multifile_download_size

Prevent multiple files with total aggregate size greater than this value in bytes from being downloaded as a zip archive.

Default: `1073741824`

property omero.web.middleware

omero.web.middleware

Warning: Only system administrators should use this feature. List of Django middleware classes in the form [{"class": "class.name", "index": FLOAT}]. See [Django middleware](#). Classes will be ordered by increasing index

Default: [{"index": 1, "class": "django.middleware.common.BrokenLinkEmailsMiddleware"}, {"index": 2, "class": "django.middleware.common.CommonMiddleware"}, {"index": 3, "class": "django.contrib.sessions.middleware.SessionMiddleware"}, {"index": 4, "class": "django.middleware.csrf.CsrfViewMiddleware"}, {"index": 5, "class": "django.contrib.messages.middleware.MessageMiddleware"}, {"index": 6, "class": "django.middleware.clickjacking.XFrameOptionsMiddleware"}]

property omero.web.nginx_server_extra_config

omero.web.nginx_server_extra_config

Extra configuration lines to add to the Nginx server block. Lines will be joined with `\n`. Remember to terminate lines with `;` when necessary.

Default: `[]`

property `omero.web.open_with`

omero.web.open_with

A list of viewers that can be used to display selected Images or other objects. Each viewer is defined as `["Name", "url", options]`. Url is `reverse(url)`. Selected objects are added to the url as `?image=1&image=2` Objects supported must be specified in options with e.g. `{"supported_objects": ["images"]}` to enable viewer for one or more images.

Default: `[["Image viewer", "\webgateway", {"supported_objects": ["image"], "script_url": "\web-client/javascript/ome.openwith_viewer.js"}]]`

property `omero.web.page_size`

omero.web.page_size

Number of images displayed within a dataset or ‘orphaned’ container to prevent from loading them all at once.

Default: `200`

property `omero.web.ping_interval`

omero.web.ping_interval

Timeout interval between ping invocations in seconds

Default: `60000`

property `omero.web.pipeline_css_compressor`

omero.web.pipeline_css_compressor

Compressor class to be applied to CSS files. If empty or `None`, CSS files won’t be compressed.

Default: `None`

property `omero.web.pipeline_js_compressor`

omero.web.pipeline_js_compressor

Compressor class to be applied to JavaScript files. If empty or None, JavaScript files won't be compressed.

Default: *None*

property `omero.web.pipeline_staticfile_storage`

omero.web.pipeline_staticfile_storage

The file storage engine to use when collecting static files with the collectstatic management command. See [the documentation](#) for more details.

Default: *pipeline.storage.PipelineStorage*

property `omero.web.plate_layout`

omero.web.plate_layout

If 'shrink', the plate will not display rows and columns before the first Well, or after the last Well. If 'trim', the plate will only show Wells from A1 to the last Well. If 'expand' (default), the plate will expand from A1 to a multiple of 12 columns x 8 rows after the last Well.

Default: *expand*

property `omero.web.prefix`

omero.web.prefix

Used as the value of the SCRIPT_NAME environment variable in any HTTP request.

Default: *None*

property `omero.web.public.cache.enabled`

omero.web.public.cache.enabled

Default: *false*

property `omero.web.public.cache.key`

omero.web.public.cache.key

Default: *omero.web.public.cache.key*

property `omero.web.public.cache.timeout`

omero.web.public.cache.timeout

Default: *86400*

property omero.web.public.enabled

omero.web.public.enabled

Enable and disable the OMERO.web public user functionality.

Default: *false*

property omero.web.public.get_only

omero.web.public.get_only

Restrict public users to GET requests only

Default: *true*

property omero.web.public.password

omero.web.public.password

Password to use during authentication.

Default: *None*

property omero.web.public.server_id

omero.web.public.server_id

Server to authenticate against.

Default: *1*

property omero.web.public.url_filter

omero.web.public.url_filter

Set a regular expression that matches URLs the public user is allowed to access. If this is not set, no URLs will be publicly available.

Default: *(?#This regular expression matches nothing)a^*

property omero.web.public.user

omero.web.public.user

Username to use during authentication.

Default: *None*

property omero.web.redirect_allowed_hosts

omero.web.redirect_allowed_hosts

If you wish to allow redirects to an external site, the domains must be listed here. For example ["openmicroscopy.org"].

Default: *[]*

property omero.web.root_application

omero.web.root_application

Override the root application label that handles /. **Warning** you must ensure the application's URLs do not conflict with other applications. omero-gallery is an example of an application that can be used for this (set to gallery)

Default: *[empty]*

property omero.web.search.default_group

omero.web.search.default_group

ID of the group to pre-select in search form. A value of 0 or -1 pre-selects All groups.

Default: *0*

property omero.web.search.default_user

omero.web.search.default_user

ID of the user to pre-select in search form. A value of 0 pre-selects the logged-in user. A value of -1 pre-selects All Users if the search is across all groups or All Members if the search is within a specific group.

Default: *0*

property omero.web.secret_key

omero.web.secret_key

A boolean that sets SECRET_KEY for a particular Django installation.

Default: *None*

property omero.web.secure

omero.web.secure

Force all backend OMERO.server connections to use SSL.

Default: *false*

property omero.web.secure_proxy_ssl_header

omero.web.secure_proxy_ssl_header

A tuple representing a HTTP header/value combination that signifies a request is secure. Example `'["HTTP_X_FORWARDED_PROTO_OMERO_WEB", "https"]'`. For more details see [secure proxy ssl header](#).

Default: `[]`

property omero.web.server_list

omero.web.server_list

A list of servers the Web client can connect to.

Default: `[["localhost", 4064, "omero"]]`

property omero.web.session_cookie_age

omero.web.session_cookie_age

The age of session cookies, in seconds.

Default: *86400*

property omero.web.session_cookie_domain

omero.web.session_cookie_domain

The domain to use for session cookies

Default: *None*

property omero.web.session_cookie_name

omero.web.session_cookie_name

The name to use for session cookies

Default: *None*

property omero.web.session_cookie_path

omero.web.session_cookie_path

The path to use for session cookies

Default: *None*

property omero.web.session_cookie_samesite

omero.web.session_cookie_samesite

The value of the SameSite flag on the session cookie. This flag prevents the cookie from being sent in cross-site requests thus preventing CSRF attacks and making some methods of stealing session cookie impossible.

Default: *Lax*

property omero.web.session_cookie_secure

omero.web.session_cookie_secure

Restrict session cookies to HTTPS only, you are strongly recommended to set this to **true** in production.

Default: *false*

property omero.web.session_engine

omero.web.session_engine

Controls where Django stores session data. See [Configuring the session engine for more details](#). Allowed values are: `omero.web.filesessionstore` (deprecated), `django.contrib.sessions.backends.db`, `django.contrib.sessions.backends.file`, `django.contrib.sessions.backends.cache` or `django.contrib.sessions.backends.cached_db`.

Default: *django.contrib.sessions.backends.file*

property omero.web.session_expire_at_browser_close

omero.web.session_expire_at_browser_close

A boolean that determines whether to expire the session when the user closes their browser. See [Django Browser-length sessions vs. persistent sessions documentation](#) for more details.

Default: *true*

property omero.web.session_serializer

omero.web.session_serializer

You can use this setting to customize the session serialization format. See [Django session serialization documentation](#) for more details.

Default: *django.contrib.sessions.serializers.PickleSerializer*

property omero.web.sharing.opengraph

omero.web.sharing.opengraph

Dictionary of *server-name: site-name*, where server-name matches a name from *omero.web.server_list*. For example: `'{"omero": "Open Microscopy"}'`

Default: `{}`

property omero.web.sharing.twitter

omero.web.sharing.twitter

Dictionary of *server-name: @twitter-site-username*, where server-name matches a name from *omero.web.server_list*. For example: `'{"omero": "@openmicroscopy"}'`

Default: `{}`

property omero.web.show_forgot_password

omero.web.show_forgot_password

Allows to hide 'Forgot password' from the login view - useful for LDAP/ActiveDir installations

Default: *true*

property omero.web.static_root

omero.web.static_root

The absolute path to the directory where collectstatic will collect static files for deployment. If the staticfiles contrib app is enabled (default) the collectstatic management command will collect static files into this directory.

Default: */home/omero/OMERO.server/var/static*

property omero.web.static_url

omero.web.static_url

URL to use when referring to static files. Example: `'/static/'` or `'http://static.example.com/'`. Used as the base path for asset definitions (the Media class) and the staticfiles app. It must end in a slash if set to a non-empty value.

Default: `/static/`

property `omero.web.staticfile_dirs`

omero.web.staticfile_dirs

Defines the additional locations the staticfiles app will traverse if the FileSystemFinder finder is enabled, e.g. if you use the `collectstatic` or `findstatic` management command or use the static file serving view.

Default: `[]`

property `omero.web.template_dirs`

omero.web.template_dirs

List of locations of the template source files, in search order. Note that these paths should use Unix-style forward slashes.

Default: `[]`

property `omero.web.thumbnails_batch`

omero.web.thumbnails_batch

Number of thumbnails retrieved to prevent from loading them all at once. Make sure the size is not too big, otherwise you may exceed limit request line, see https://docs.gunicorn.org/en/latest/settings.html?highlight=limit_request_line

Default: `50`

property `omero.web.time_zone`

omero.web.time_zone

Time zone for this installation. Choices can be found in the TZ database name column of: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones Default `"Europe/London"`

Default: `Europe/London`

property `omero.web.top_logo`

omero.web.top_logo

Customize the webclient top bar logo. The recommended image height is 23 pixels and it must be hosted outside of OMERO.web.

Default: *[empty]*

property omero.web.top_logo_link

omero.web.top_logo_link

The target location of the webclient top logo, default unlinked.

Default: *[empty]*

property omero.web.ui.center_plugins

omero.web.ui.center_plugins

Add plugins to the center panels. Plugins are ['Channel overlay', 'webtest/webclient_plugins/center_plugin.overlay.js.html', 'channel_overlay_panel']. The javascript loads data into \$('#div_id').

Default: *[]*

property omero.web.ui.metadata_panes

omero.web.ui.metadata_panes

Manage Metadata pane accordion. This functionality is limited to the existing sections.

Default: *[{"name": "tag", "label": "Tags", "index": 1}, {"name": "map", "label": "Key-Value Pairs", "index": 2}, {"name": "table", "label": "Tables", "index": 3}, {"name": "file", "label": "Attachments", "index": 4}, {"name": "comment", "label": "Comments", "index": 5}, {"name": "rating", "label": "Ratings", "index": 6}, {"name": "other", "label": "Others", "index": 7}]*

property omero.web.ui.right_plugins

omero.web.ui.right_plugins

Add plugins to the right-hand panel. Plugins are ['Label', 'include.js', 'div_id']. The javascript loads data into \$('#div_id').

Default: *[["Acquisition", "webclient/data/includes/right_plugin.acquisition.js.html", "meta-data_tab"], ["Preview", "webclient/data/includes/right_plugin.preview.js.html", "preview_tab"]]*

property omero.web.ui.top_links

omero.web.ui.top_links

Add links to the top header: links are ['Link Text', 'link|lookup_view', options], where the url is reverse('link'), simply 'link' (for external urls) or lookup_view is a detailed dictionary {"viewname": "str", "args": [], "query_string": {"param": "value"}}, E.g. ["Webtest", "webtest_index"] or ["Homepage", "http://...", {"title": "Homepage", "target": "new"}] or ["Repository", {"viewname": "webindex", "query_string": {"experimenter": -1}}, {"title": "Repo"}]

Default: `[["Data", "webindex", {"title": "Browse Data via Projects, Tags etc"}], ["History", "history", {"title": "History"}], ["Help", "https://help.openmicroscopy.org", {"title": "Open OMERO user guide in a new tab", "target": "new"}]]`

property omero.web.use_x_forwarded_host

omero.web.use_x_forwarded_host

Specifies whether to use the X-Forwarded-Host header in preference to the Host header. This should only be enabled if a proxy which sets this header is in use.

Default: *false*

property omero.web.user_dropdown

omero.web.user_dropdown

Whether or not to include a user dropdown in the base template. Particularly useful when used in combination with the OMERO.web public user where logging in may not make sense.

Default: *true*

property omero.web.viewer.view

omero.web.viewer.view

Django view which handles display of, or redirection to, the desired full image viewer.

Default: *omeroweb.webclient.views.image_viewer*

property omero.web.webgateway_cache

omero.web.webgateway_cache

Default: *None*

property omero.web.wsgi_args

omero.web.wsgi_args

A string representing Gunicorn additional arguments. Check Gunicorn Documentation <https://docs.gunicorn.org/en/latest/settings.html>

Default: *None*

property omero.web.wsgi_timeout

omero.web.wsgi_timeout

Workers silent for more than this many seconds are killed and restarted. Check Gunicorn Documentation <https://docs.gunicorn.org/en/stable/settings.html#timeout>

Default: *60*

property omero.web.wsgi_workers

omero.web.wsgi_workers

The number of worker processes for handling requests. Check Gunicorn Documentation <https://docs.gunicorn.org/en/stable/settings.html#workers>

Default: *5*

property omero.web.x_frame_options

omero.web.x_frame_options

Whether to allow OMERO.web to be loaded in a frame.

Default: *SAMEORIGIN*

2.5.8 Syslog configuration

syslog is a standard for message logging over networks. OMERO.server supports logging to either a local or remote syslog service.

This allows all logs of the OMERO.server to be routed to a central location instead of (or as well as) to a file.

Note: It is important to note that this applies only to the OMERO.server itself, not to components like OMERO.web.

How it works

Whenever a log message is generated, OMERO's logging framework will forward that message to any configured appenders.

By default, OMERO is configured to log everything to files.

Note: OMERO is configured to log a record of events for operations such as import. These are written directly to the Managed Repository. It is very likely that even if replacing file logging with syslog, this aspect should be retained in files. This is easily achieved by not changing any loggers using *SIFT*.

Configuration

To configure OMERO to be able to log to syslog, it is necessary to modify the file `OMERO.server/current/etc/logback.xml`. It is possible to do all the configuration changes in this file alone, but for ease of config management, it is demonstrated here where an additional `OMERO.server/current/etc/logback_syslog.xml` file is used in addition.

The following information is required to configure OMERO to log to syslog.

- The host on which syslog is running: e.g. *localhost*
- The port number on which syslog is running on that host: e.g. *514*
- The facility (RFC 3164) that OMERO should be handled as: e.g. *user* or *local6*

Note: The facility is important because it determines how syslog will handle the messages it receives. It is unlikely that OMERO's log output will be desired in a local systems primary message log for example. On Linux this is often `/var/log/messages`. Remember to configure the syslog configuration to avoid this. This is also where configuration of onward forwarding can be configured (to a service such as [splunk](#)). Finally, syslog can be configured to specifically output this facility output to a file such as `/var/log/omero`.

Create the new file `OMERO.server/current/etc/logback_syslog.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<included>
  <!-- syslog -->
  <appender name="SYSLOG" class="ch.qos.logback.classic.net.SyslogAppender">

    <!-- Exclude debug level logging from ome.services.blitz.repo.ManagedImportRequestI -
    ->
    <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
      <evaluator> <!-- defaults to type ch.qos.logback.classic.boolex.
    JaninoEventEvaluator -->
      <expression>return Level.DEBUG.equals(Level.toLevel(level)) && logger.
    equals("ome.services.blitz.repo.ManagedImportRequestI");</expression>
    </evaluator>
    <OnMismatch>NEUTRAL</OnMismatch>
    <OnMatch>DENY</OnMatch>
    </filter>
    <!-- Exclude debug level logging from omero.* (except allow omero.cmd.*) -->
    <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
      <evaluator> <!-- defaults to type ch.qos.logback.classic.boolex.
```

(continues on next page)

(continued from previous page)

```

↪ JaninoEventEvaluator -->
    <expression>return Level.DEBUG.equals(Level.toLevel(level)) && logger.
↪ startsWith("omero.") && !logger.startsWith("omero.cmd.");</expression>
    </evaluator>
    <OnMismatch>NEUTRAL</OnMismatch>
    <OnMatch>DENY</OnMatch>
</filter>

<syslogHost>localhost</syslogHost>
<facility>local6</facility>
<suffixPattern>OMERO [%level] [%thread] %logger %msg</suffixPattern>
</appender>

</included>

```

This creates an appender that sends messages to syslog. *syslogHost* is the host on which syslog is running. No port is specified as *514* is the default. The *suffixPattern* is customizable. In this instance it is identical to OMERO's file logger except an added "OMERO" identifier has been added for clarity. The name of the appender has been set to *SYSLOG*. The filters replicate the same behaviour from the default *FILE* appender.

Note: If configuring the appender directly in the `OMERO.server/current/etc/logback.xml` file, then the *included* tag should not be used.

Within the *configuration* tag of `OMERO.server/current/etc/logback.xml` add:

```
<include file="/path/to/OMERO.server/etc/logback_syslog.xml"/>
```

Note: The included file path can be relative, but note that it is NOT relative to the `OMERO.server/current/etc/logback.xml` file, but to the current directory set by OMERO. It is highly recommended to use a full path.

Finally, also within `OMERO.server/current/etc/logback.xml` modify the *root* tag to include a second *appender-ref* (It can also be replaced if the file logs are not desired or syslog will handle writing those to a file on OMERO's behalf):

```

<root level="OFF">
  <appender-ref ref="SYSLOG"/>
  <appender-ref ref="FILE"/>
</root>

```

Note: A restart of OMERO will be necessary before this takes effect.

2.6 Managing OMERO

This section contains details on how to manage users, groups and data access in OMERO. New in OMERO 5.4.0, full administrators can now create restricted administrators to allow facility managers or other trusted users to carry out tasks on behalf of all users.

2.6.1 Groups and permissions system

See also:

OMERO permissions querying, usage and history

Summary

A user may belong to one or more groups, and the data in a group may **at most** be shared with users in the same group on the same OMERO server. The degree to which their data is available to other members of the group depends on the permissions settings for that group. Whenever a user logs on to an OMERO server, they are connected under one of their groups. All data they import and any work that is done is assigned to the current group, however the user can move their data into another group.

Users

Administrator

Your OMERO server will have one or more administrators. Each group can be administrated by any of your server administrators. The administrators control all settings for groups.

Group owner

Your group may have one or more owners. The group owner has some additional rights within each group compared to a standard group member, including the ability to add other members to the group.

Group member

This is the standard user.

Restricted Administrators

New in OMERO 5.4.0, these administrators can be created with a subset of privileges allowing trusted users to act on behalf of all other OMERO users for a defined set of tasks. See *Administrators with restricted privileges* for further information.

Groups and users must be created by the server administrator or a restricted administrator with the correct privileges. Users can then be added by the administrator (either a full admin or a restricted admin with the correct privileges) or by one of the group owners assigned by the administrator (group owners would typically include the PI of the lab). The group's owners or administrators can also choose the permission level for that group. See the [User guides for managing groups and users](#) for more information about how to administrate them in OMERO.

Group permission levels

The various permission levels are:

Private

This group is the most restrictive:

- A private *Group owner* can see and control who the group members are and can view their data.
- As a *Group member*, you will only ever be able to see your own data.
- This can be used for general data storage, access and analysis, but has very limited collaboration potential other than for the *Group owner* to see other group members' data.

Potential use cases of Private group:

- A PI as *Group owner* and their student, as a *Group member*, can access the student's data. A student might use this to store all of their data and from here, the PI and/or student might decide which data could/should be moved into a more collaborative group where additional members would also be able to view the data.
- An institutional repository type structure where data are being archived, but not necessarily open for general viewing.

Read-only

This group allows visibility of other users and their data, but minimal ability to annotate their data:

- The *Group owner* can control group members as above and can perform annotations on the other group members data.
- *Group member* can see who other members are and view their data, but cannot annotate another members' data at all.

Potential use cases of Read-only group:

- A scientist might move data into a read-only group when they want other group members to access and view their data. Their PI, as a group owner could then annotate and/or add Regions of Interest (ROIs) to their images.
- Scientists submitting a publication could move data to a read-only group as part of the publication workflow, making them publicly available via a URL for reviewers and readers (see the [Users guide for public data](#)).
- For an institutional repository where data are being archived and then available for other users in the institute to view; this could be standard storage of all original data, or for data that is included in publications.

Read-annotate

This group allows some collaboration on other members' data for all members:

- *Group member* can view other members, their data and can make annotations on those other members' data.

Potential use cases of Read-annotate group:

- This could be used by a group of scientists working together with data for a publication.

Read-write

This group essentially allows all the group members to behave as if they co-own all the data:

- *Group member* can view, annotate, edit and **delete** all data; the only restriction is that they cannot move other members' data into another group.

Potential use cases of Read-write group:

- A group of scientists working in a completely collaborative way, trusting every member of the group to have equal rights and access to all the data.

Note: Restricted administrators are designed to work independently of group permissions. They act as full administrators when using their subset of privileges, allowing them to perform actions on data belonging to other users even in private groups (see the permissions tables below).

See also:

Users guide for data management and sharing

Workflow guide describing the basics of data management and sharing with links to more detailed permissions documentation

Changing group permissions

It is possible for the *Group owner* or server *Administrator* to change the permissions level on a group after it has been created and filled with data, with the following limitations:

- It is not possible to ‘reduce’ permissions to *Private* if the group contains a projection made by one member from data owned by another user. In other circumstances, reducing permissions to private will warn of loss of annotations etc. as noted below, but will still be possible.
- Only *Administrator* can promote a group to *Read-write* permissions. **Make certain all the members understand that this allows anyone in the group to permanently delete any of the data before performing this action.**

Warning: Please be very careful before downgrading a group’s permission level. If a user has annotated other users’ data and the group is downgraded, any links to annotations that are not permitted by the new permissions level will be lost.

Permissions on your and other users’ data

What can you do with your data?

All OMERO users in all groups can perform all actions on their own data (with the exception of changing the ownership of the data).

The main actions available include, but are not limited to:

- create projects and/or datasets
- import data
- delete data
- edit names and descriptions of images
- change rendering settings on images
- annotate images (rate, tag, add attachments and comments)
- de-annotate (remove annotations that you have added)
- use Regions of Interest (ROIs) (add, import, edit, delete, save and analyze them)
- run scripts
- move data between groups, if you belong to more than one group

What can you do with someone else's data in your group?

Actions available for you on someone else in your group's data will depend both on the permissions of the group you are working in, and what sort of user you are. See the table below for a quick reference guide to permissions available on other people's data.

Some of these policies may evolve as the permissions functionality matures in response to user feedback. Please let us know any comments or suggestions you have via our [mailing lists](#) or [forums](#).

Permissions tables

The following are the permissions valid for users working on data belonging to other group members. These permissions depend on the group permissions and on the type of the user performing the action.

Restricted administrators act as full administrators when using their subset of privileges. For all actions which are not covered by their privileges subset, they act as standard group members. For example, a data analyst with write data privileges can edit data even in a private group (without having to be a member of that group) but without the delete privilege they cannot delete data belonging to another user unless that data is in a read-write group they are a member of. All restricted administrators can view and download any data regardless of group type and their subset of privileges. See *Administrators with restricted privileges* for further information.

Administrator

This table covers both full server administrators and restricted administrators with the privileges required for these actions. Restricted administrators act as group members for any actions that are not covered by their subset of privileges.

<i>Action</i>	<i>Private</i>	<i>Read-only</i>	<i>Read-annotate</i>	<i>Read-write</i>
<i>View</i>	Y	Y	Y	Y
<i>Annotate</i>	N	Y	Y	Y
<i>Delete</i>	Y	Y	Y	Y
<i>Edit</i>	Y	Y	Y	Y
<i>Move between groups</i>	Y	Y	Y	Y
<i>Remove annotations</i>	Y	Y	Y	Y
<i>Mix data</i>	N	Y	Y	Y
<i>Change ownership</i>	Y	Y	Y	Y

Group owner

<i>Action</i>	<i>Private</i>	<i>Read-only</i>	<i>Read-annotate</i>	<i>Read-write</i>
<i>View</i>	Y	Y	Y	Y
<i>Annotate</i>	N	Y	Y	Y
<i>Delete</i>	Y	Y	Y	Y
<i>Edit</i>	Y	Y	Y	Y
<i>Move between groups</i>	N	N	N	N
<i>Remove annotations</i>	Y	Y	Y	Y
<i>Mix data</i>	N	Y	Y	Y
<i>Change ownership</i>	Y	Y	Y	Y

Group member

<i>Action</i>	<i>Private</i>	<i>Read-only</i>	<i>Read-annotate</i>	<i>Read-write</i>
<i>View</i>	N	Y	Y	Y
<i>Annotate</i>	N	N	Y	Y
<i>Delete</i>	N	N	N	Y
<i>Edit</i>	N	N	N	Y
<i>Move between groups</i>	N	N	N	N
<i>Remove annotations</i>	N	N	N	Y
<i>Mix data</i>	N	N	N	Y
<i>Change ownership</i>	N	N	N	N

Key

Action

Action on other users' data.

Annotate

Add annotations (rating, tag, attachment, comment, ROI) to another users' data. Also create & save ROIs (save ROIs that you draw on another users' data).

Change ownership

Assign ownership of the data to a different user. The target user should be a member of the group the data belongs to.

Delete

Delete data such as images or ROIs. ROIs may have been added by others or yourself.

Edit

Modify the name or description of other users' objects such as images.

Mix data

Copy, Move or Remove other users' data to or from your Projects, Datasets or Screens. Copy, Move or Remove your or others' data to or from others' Projects, Datasets or Screens.

Note: You should always be able to remove annotations (such as tags) that you linked to other users' data (you own the link). The link can be deleted, but the tag itself will not be deleted.

Move between groups

Only an admin has the right to move other users' data between groups.

Note: An admin does not have to be a member of either the original or the destination group.

Remove annotations

Remove annotations made by others on your data.

Render

Create your own rendering settings (this will not modify the settings of the owner).

View

View other users' data such as images. View ROIs added by others. Draw ROIs on other users' data, but they cannot be saved.

Issues to be aware of

ROIs

- You can never edit (change text or move) other users' ROI.
- Any ROIs added to other users' data will not affect ROIs added by the owner.

Tags and attachments

- A tag or attachment is 'owned' by the person who creates it or uploads it to the server.
- The link between a tag or an attachment is 'owned' by the person who annotates an image with that tag or attachment i.e. makes a link between the tag/attachment and the image.
- De-annotation deletes the link between the tag/attachment and image but does not remove/delete the tag or attachment from the system.

Scripts

- Although all users can run scripts on other users' data, the actions within those scripts will be subject to the restrictions of the permissions detailed in the tables above.

2.6.2 Administrators with restricted privileges

Summary

OMERO allows you to create administrators with a subset of the full administrator privileges. This is a way to cater for the need for more powerful users acting on behalf of all other OMERO users, with no group membership but with access to all groups and data of all users in OMERO. This should be achieved without creating new full administrators in OMERO. In the real world, these administrators with restricted privileges (restricted admins) will typically be imaging facility managers, image analysts, or anybody who needs to organize users and data of others in OMERO. Even a restricted administrator is still a powerful user so each must be a highly trusted individual.

Warning: Restrictions on privileges can communicate an administrator's intended role and prevent many accidents whose consequences would be severe. However, even a little privilege can go a long way so never grant administrative powers lightly. These restrictions do *not* protect against a cunning, malicious user: never elevate a user even as far as group ownership unless they are truly trusted.

Full administrators in OMERO can create new administrators with restricted privileges using the OMERO.web interface, see the [Web Interface: Managing Users](#) in our User guides. OMERO.cli does not yet support easy management of restrictions nor does it offer the helpful *permissions mapping* but advanced users may *use OMERO.cli to adjust the restrictions* on an administrator.

Four suggested workflows

We suggest here four setups that should cover the four mainstream workflows. Nevertheless, you can combine the privileges (check the checkboxes in the OMERO.web interface) in any way you see fit. The privileges were designed in such a way that they still bear useful functionality even when used in isolation. For example, checking the *Chown* checkbox will give the new administrator with restricted privileges the power to transfer ownership of other users' data. For exact server-side definitions of the privileges displayed in OMERO.web interface see [Administrator restrictions: relating OMERO.webadmin to OMERO.server](#).

Required Privileges	<i>Data Viewer</i>	<i>Importer</i>	<i>Analyst</i>	<i>Group and Data Organizer</i>
<i>Sudo</i>	N	Y	N	N
<i>Write Data</i>	N	N	Y	Y
<i>Delete Data</i>	N	N	N	Y
<i>Chgrp</i>	N	N	N	Y
<i>Chown</i>	N	N	Y (O)	Y
<i>Create and Edit Groups</i>	N	N	N	Y
<i>Create and Edit Users</i>	N	N	N	Y
<i>Add Users to Groups</i>	N	N	N	Y
<i>Upload Scripts</i>	N	N	Y	N

Y

privilege required, checkbox in OMERO.web interface is checked

N

privilege not required, checkbox is not checked

O

privilege optional for the workflow

Note: Restricted admins workflows in OMERO.clients

Please do not expect for any workflows mentioned here that all OMERO.clients OMERO.web, OMERO.insight, command line interface (CLI) are fully equipped to execute them (see details below). New features will be added in OMERO.clients in the 5.4.x series of OMERO releases.

Note: Group membership

All the workflows here assume that the administrator with restricted privileges is not a member of any group except the System group. This does not preclude such administrator from being a member of any number of groups. Inside the groups the restricted admin is a member of, they have the same privileges as other group members of that group additionally to their administrative privileges.

Note: Deleting privileges

Sudo privilege includes ability to delete the data of the user whom the administrator is working on behalf of. If you want to prevent the restricted admin from deleting others' data entirely, do not give *Delete Data* and do not give *Sudo* privileges.

Note: Privilege escalation

The administrators with restricted privileges (restricted admins) are prevented from escalation of their privileges. Creation of a restricted admin with higher privileges than the creator, and creation of a full administrator, are prevented. Furthermore, although a restricted admin can Sudo on behalf of a full administrator, their privileges will not expand to the full administrator privilege set by this action. See also *Sudo*.

Workflow 1: Data Viewer

If you do not give any explicit privileges to the administrator with restricted privileges, this administrator still has some useful privileges. These include browsing and viewing all the data of all users in all groups (including the groups where they are not members). The administrator with restricted privileges is also able to download all the data in all types of groups. Furthermore, they can view user and group information, such as usernames, e-mail addresses, group permission levels and lists of all users and groups. They are not able to annotate, edit or delete any of the data or change any user or group information though. Note that any administrator with restricted privileges described below or otherwise created combining the privileges at will would be able to perform the Data Viewer workflow as well.

Client Details:

- OMERO.insight: is not designed to show any groups, or data belonging to any groups, you are not a member of. The Data Viewer workflow is preferably executed using OMERO.web or CLI.
- OMERO.web: allows viewing and downloading the data, see [Viewing Data](#) and [Downloading Data](#)
- CLI: allows listing all images, groups and users and downloading the data:

```
# List all users on server
$ omero user list
# List all groups on server
```

(continues on next page)

(continued from previous page)

```
$ omero group list
# List all images on server
$ omero fs images
```

Workflow 2: Importer

The Importer role is to import images into OMERO for other users, i.e. in such a manner that the imported images are owned by the users in OMERO, not by the user in the role of the Importer. The Importer role is typically used by an imaging facility manager who is importing data acquired by users on microscopes into OMERO.

The importer workflow can be achieved with only the *Sudo* privilege (first line in the above table). This privilege allows them to “become” the user they are importing the data for. The Importer role may need to reorganize the imported data. For example, they made a mistake, Sudoed as a wrong user in a wrong group and need to rectify the mistake using the command line interface (CLI) client. Whilst being sudoed, the Importer role can Delete the wrongly imported data (even without Delete privilege given, see the Note above), logout, login and *Sudo* as the correct user and repeat the import process. In short, whilst Sudoed, Importer role can do any action which the user they are becoming is allowed to do. In case any more post-import cleaning and data organizing is necessary for Importer, this might be enabled by giving them also privilege necessary for the Data organizers (see *Workflow 4: Group and Data Organizer* below).

If you have any doubts about giving the administrators with restricted privilege the *Sudo* privilege (which implicitly gives the ability to delete other users’ data), there are two workarounds which enable import for others without *Sudo*.

The first, simpler, workaround involves importing the data as Importer into the group of the future data owner and then transferring the ownership of the data (see details in *Workflow 3: Analyst*). The second workaround involves importing into the group of the Importer as the importer, then moving the data into the group of the prospective data owner and then changing the ownership of the data to the owner (necessary tools are described in *Workflow 3: Analyst*).

Client details:

- OMERO.importer or OMERO.insight: you have to be a member of the group you want to import to in OMERO.importer or OMERO.insight. Login as the administrator with restricted privileges and perform the import for others as described in the chapter of the User guides *import for others*.
- CLI: documentation is available covering *Import images* and *Import targets* (see also the videos on import on the *OME YouTube channel*):

```
# Login as the Importer and sudo as the user you want to import for
$ omero --sudo Importer -u user login
# Create new containers belonging to the user
$ omero obj new Dataset name=Dataset-of-user
$ omero obj new Project name=Project-of-user
# Link the containers
$ omero obj new ProjectDatasetLink parent=Project:17 child=Dataset:13
# Import into created Dataset
$ omero import ~/Desktop/CMP01.png -T Dataset:name:Dataset-of-user
```

Workflow 3: Analyst

Typically, the Analyst role in OMERO is to

- read the data (always possible, see [Workflow 1: Data Viewer: Data Viewer](#))
- change and save the rendering settings of the images (enabled by [Write Data](#) privilege, exception is Private groups, where they cannot save rendering settings)
- annotate the data (enabled by [Write Data](#) privilege, but not possible in Private groups)
- draw and save ROIs on other users' images (enabled by [Write Data](#) privilege, but no saving in Private groups possible)
- upload and attach result files to the analyzed images (enabled by [Write Data](#) privilege, except Private groups, where attaching is not possible)
- create Projects and Datasets for newly imported images in groups they are not a member of (enabled by [Write Data](#) privilege)
- import new images resulting from image analysis into these Projects and Datasets
- link new images resulting from image analysis to existing Projects and Datasets of the original data owner (enabled by [Write Data](#) privilege)
- (possibly) changing the ownership of the newly created containers and contained result images to the users (enabled by [Chown](#) privilege)
- upload, edit and delete official scripts usable by all OMERO users (enabled by [Upload Scripts](#) privilege)

Client details:

- OMERO.insight or Insight-ImageJ plugin: Analyst has to be a member of the group where the data is located. They can draw ROIs and extract analysis results from the ROIs and data in any type of group. They can save ROIs except in Private groups. They can upload official scripts in OMERO.insight (any group type, Analyst does not have to be a member of any particular group for script upload in OMERO.insight).
- OMERO.web, OMERO.insight, Insight-ImageJ plugin: Analyst can adjust rendering settings and save them, upload attachments with results and annotate (for example tag, key-value pairs, rating, commenting). These actions are not permitted in Private groups with images belonging to others. See [User guides example](#).
- CLI: Upload of official scripts is allowed (in any group type, see [OMERO.scripts user guide](#) and below). Upload of attachments with results, annotating (not in private group), creating containers, import of resulting images into groups you are not a member of (in private groups these are invisible for the owner of the original data, unless you transfer their ownership), transferring ownership of these containers (any group type), transferring ownership of objects (images, annotations, ROIs, uploaded attachments with results) is possible too (see [Command Line Interface as an OMERO client](#)):

```
# Upload an official script
$ omero script upload --official /PATH/TO/YOUR_SCRIPT
# Login to the group the original data are in
$ omero -g testgroup login
# Create new Dataset
$ omero obj new Dataset name=new-dataset
# Import result images into the Dataset
$ omero import -T Dataset:name=new-dataset /PATH/TO/RESULT/IMAGES
# Transfer the ownership of the Dataset and
# of the contained images to the user with ID:55
$ omero chown 55 Dataset:112
```

Workflow 4: Group and Data Organizer

Group and Data Organizer role is for creation of new users and groups in OMERO and allocating the users to appropriate groups. It is also possible to change the users' information such as e-mail and to change group permissions level. These tasks are facilitated by the privileges *Create and Edit Groups*, *Create and Edit Users* and *Add Users to Groups*.

The Group and Data Organizer might also be tasked with dealing with data owned by OMERO users who have left the institution. The Organizer can transfer ownership of the data owned by the leaving person (facilitated by the *Chown* privilege) to another user. In cases where the new owner of the data may not be a member of the data group, the Organizer first moves the data between groups (facilitated by the *Chgrp* privilege), and then transfers the ownership of the data. Always try to avoid the situation where owner of the data is not in data group.

For moving data between groups, usage of OMERO.web is highly recommended. The Organizer can create new containers (Projects, Datasets) on behalf of data owner in OMERO.web conveniently as part of the Move to Group command in OMERO.web (*Move to Group*). The containers and links of data to containers will belong to data owner. For new container creation and linking, the *Write Data* privilege is necessary. CLI can be used for the move action as well, see *Moving objects between groups*.

In case of data owner not being in the group where the data is, the Organizer can also add the data owner to the data group (facilitated by the *Add Users to Groups* privilege), instead of moving the data. The Organizer will transfer the ownership of the data to the new owner only after they have added the new data owner to the data group.

During all data manipulation steps, the Organizer needs the *Write Data* privilege to create new Projects, Datasets or Screens for the new owners of the data and to link the data to those containers or to already existing containers owned by the new owner. Since OMERO 5.4.0, OMERO.web enables Organizers with *Write Data* privilege to create new containers belonging to other users, see the *Manage Images in Datasets/Projects* section of our User guides. Except the links created during creation of new Datasets inside others' Projects in OMERO.web, any links created by the Organizer will belong to the Organizer, not the owner of the data. This will be addressed in OMERO.web in the 5.4.x series. The ownership transfer of the containers and links can be done later on the CLI. Linking of others' data is never possible in Private groups.

After the Organizer has dealt with the data, they can remove the leaving person from any group (included in the *Add Users to Groups* privilege) and make the user inactive (facilitated by the *Create and Edit Users* privilege).

Note that the ownership of data of a user can be transferred either piecemeal, i.e. specifying each Project or Dataset to transfer (using `omero chown` command of CLI), or all of the data of the user can be transferred in one step. The transfer of all the data of the user in one step has to be considered an advanced feature; it may be slow and demanding of CPU resources in cases of complex data.

Quite naturally the Group and Data Organizer can be easily split into two separate roles, with the Group Organiser role having *Create and Edit Groups*, *Create and Edit Users*, *Add Users to Groups* privileges, and the Data Organiser role having *Write Data*, *Delete Data*, *Chgrp*, *Chown* privileges. It is of course possible to use any combination of these privileges as you see fit. It is recommended to always grant *Create and Edit Users* with *Add Users to Groups* so that the new restricted administrator is able to deactivate users.

Client Details:

- OMERO.web: all the Data Organizing actions are possible, except transfer of ownership (possible only in CLI, will be addressed in the 5.4.x series). Creation of Projects, Datasets or Screens for other users in OMERO.web is possible since OMERO 5.4.0, see the *Manage Images in Datasets/Projects*. All the Group and User Organizing actions are possible if all *Create and Edit Groups*, *Create and Edit Users* and *Add Users to Groups* privileges are given. It is also reasonable to give *Create and Edit Users* and *Add Users to Groups* or *Create and Edit Groups* and *Add Users to Groups*. These combinations give the restricted administrator good user interface experience in OMERO.web.
- CLI: see *User/group management*, *Moving objects between groups*, *Changing ownership of objects* and examples below for CLI features useful for Group and Data Organizing:

```
# Create new user and put them into 2 groups
$ omero user add username firstname lastname group1 group2
# Edit login name of a user with ID:55
$ omero obj update Experimenter:55 omeName=new-login-name
# Add a user to a group named "testgroup"
$ omero group adduser --name testgroup --user-name newbieingroup
$ omero group removeuser --name testgroup --user-name thegoner
# Make a user a group owner. Works also when the owner-to-be
# is already a member of the group
$ omero group adduser --name group --user-name ownertobe --as-owner
# Remove a group owner from ownership of the group. Does not remove
# the formerowner from group, just unsets the ownership.
$ omero user leavegroup testgroup --name formerowner --as-owner
# Move a Dataset hierarchy to group 5 and include all annotations
# on the Dataset and objects linked to the Dataset
$ omero chgrp 5 Dataset:51 --include Annotation
# Transfer ownership to user 55 of the Project 112
$ omero chown 55 Project:112
# Transfer the ownership of a Project-Dataset link. Useful in case the
# link was created by the Organizer and links objects of others
$ omero chown 55 ProjectDatasetLink:123
# Transfer the ownership of Dataset-Image link
$ omero chown 55 DatasetImageLink:154
# Transfer all data of user 5 to user 11 (advanced, might be slow)
$ omero chown 11 Experimenter:5
```

Key

Add Users to Groups

Administrator can add or remove users to groups. See [Workflow 4: Group and Data Organizer](#) for more details.

Analyst

Administrator who performs image analysis on others' images in OMERO. See more details in [Workflow 3: Analyst](#).

Chgrp

Administrator can move others' data to a different Group. See [Workflow 4: Group and Data Organizer](#) for more details.

Chown

Administrator can transfer others' data to a different Owner. See [Workflow 4: Group and Data Organizer](#) for more details.

Create and Edit Groups

Administrator can create and edit groups (but not add or remove users). See [Workflow 4: Group and Data Organizer](#) for more details.

Create and Edit Users

Administrator can create and edit other users (but not add them to groups). See [Workflow 4: Group and Data Organizer](#) for more details.

Data Viewer

Administrator who views and downloads data of others. See more details in [Workflow 1: Data Viewer](#).

Delete Data

Administrator can delete other users' data. See Note on Delete for more details. Integral part of [Workflow 4: Group and Data Organizer](#).

Group and Data Organizer

Administrator who creates new users and groups in OMERO and allocates or removes the users to or from appropriate groups. This administrator also deals with data left after OMERO users which left the institution, or otherwise is tasked with reorganizing of others' data. See more details in [Workflow 4: Group and Data Organizer](#).

Importer

Administrator who imports images into OMERO for other users. The imported images are owned by the users in OMERO, not by the Importer. This is typically an imaging facility manager who is importing data acquired by users on microscopes into OMERO. See more details in [Workflow 2: Importer](#).

Sudo

Administrator can log in as another user, with all the permissions of that user. When the restricted admin is working on behalf of a user and using Sudo, their privileges are a common least denominator of the privileges of the user and of the restricted admin (i.e. if a restricted administrator is using Sudo on behalf of a full administrator, they do not have full admin rights to perform actions not covered by their own privileges). See also Note on privilege escalation, Note on Delete and [Workflow 2: Importer](#) for more details.

Upload Scripts

Administrator can upload "official" OMERO.scripts to the server. See [Workflow 3: Analyst](#) for more details.

Write Data

Administrator can create data in groups of which he/she is not a member. Also allows annotating, adding attachments to and editing and linking of other users' data. See [Workflow 3: Analyst](#) for more details.

Administrator restrictions: relating OMERO.webadmin to OMERO.server

Summary

OMERO allows you to create administrators with a subset of the full administrator privileges, see [Administrators with restricted privileges](#). The OMERO.web user interface form for creation and editing of restricted administrators (see the [Web Interface: Managing Users](#) section) collates the server-side privileges into fewer options and gives the options user-friendly names. Here, the mapping of the OMERO.web options to the server-side privileges is given. The server-side privileges are more granular and direct work with them is possible on the CLI, as described in [Adjusting administrator restrictions](#).

Map of the OMERO.web UI options to the server-side privileges

Option in OMERO.web	Server-side privilege(s)
Sudo	Sudo
Write data	WriteOwned, WriteFile, WriteManagedRepo
Delete data	DeleteOwned, DeleteFile, DeleteManagedRepo
Chgrp	Chgrp
Chown	Chown
Create and Edit groups	ModifyGroup
Create and Edit Users	ModifyUser
Add Users to Groups	ModifyGroupMembership
Upload Scripts	WriteScriptRepo, DeleteScriptRepo

Note: CLI lists restrictions, OMERO.web lists privileges The lists shown using CLI commands recommended in *Adjusting administrator restrictions* will be complementary lists to the ones which can be deduced from the table above.

Note: ReadSession privilege is never given to restricted admin In OMERO.web, you can never create an administrator with restricted privileges who has the “ReadSession” privilege.

See also:

- Command Line Interface guides for *User/group management* and *Changing ownership of objects*
- [Wlakthrough for Facility Managers](#)

2.7 Data Import and Storage

This section contains details of how OMERO.fs allows you to import and store data with OMERO 5.

2.7.1 OMERO.dropbox

DropBox was originally designed as the first stage of the file system changes referred to as *OMERO.fs*. It utilizes a file system monitor to find newly uploaded files and run a fully automatic import on those files if possible. This release of OMERO.dropbox runs on the same machine as the OMERO.server and watches designated areas of the local filesystem for new or modified files. If those files are importable, then an automatic import is initiated. OMERO.dropbox is started automatically when the OMERO.server starts and it will run if the prerequisites below are met.

Prerequisites

In addition to the general *System requirements* OMERO.dropbox has the following more specific requirements:

- OMERO.dropbox is built on underlying OS file-notification system, and so is only available for specific versions of certain operating systems. OMERO.dropbox has been tested on the following systems:
 - Linux with kernel 2.6.13 and higher.
 - Mac OS 10.6 and later.
- In addition some platforms require further Python packages to be available:
 - Mac OS systems that use a macports install of Python will need to have FSEvents available in the PYTHONPATH. This will require a path of the form `/System/Library/Frameworks/Python.framework/Versions/3.X/Extras/lib/python/PyObjC/` to be added, according to the version of Python used.
- The filesystem which OMERO.dropbox watches must be local to the given operating system. Watching a network-attached share (NAS) is strictly ***not*** supported.

Installing DropBox

From the OMERO 5.6.0 release, the library `omero-dropbox` supports Python 3 and is now available on [PyPI](#). We recommend you use a Python virtual environment to install it. It should be installed in the same virtual environment where `omero-py` is installed. See [OMERO.server installation](#).

Activate the environment `/opt/omero/server/venv3` where `omero-py` is installed and install `omero-dropbox` as **root**:

```
$ . /opt/omero/server/venv3/bin/activate
$ pip install omero-dropbox==5.6.2
```

Enable DropBox as the **omero-server system user** (`su - omero-server`):

```
$ omero admin ice server enable MonitorServer
$ omero admin ice server enable DropBox
```

Using DropBox

In its default configuration the monitored area of the file system is a DropBox subdirectory of the `OmeroBinaryRepository` directory. The system administrator should create DropBox and then under that a directory for each user, using their `omero` username. The ownership and permissions should be set so that a user can copy files into their DropBox directory:

```
/OMERO/DropBox/amy
                  /emily
                  /edgar
                  /root
                  /zak
```

Experimenters can add subdirectories under their named directory for convenience. Copying or moving a file of an importable file type into a named directory or nested subdirectory will initiate an automatic import of that file for that user. Multi-file formats will be imported after the last required file of a set is copied into the directory. Images and plates will be imported into the default group of the user, with images placed into `Orphaned` images unless the `target` option was configured (see below and [Import targets](#)).

Acquisition systems can then be configured to drop a user's images into a given DropBox.

Note:

- The DropBox system is designed for image files to be copied in at normal acquisition rates. Copying many files en masse may result in files failing to import.
- It is also intended as a write-once system. Modifying an image after it has been imported may result in that modified image also being imported depending on the operating system and how the image was modified.
- Once directories are created within DropBox or files are copied or moved into DropBox they should not be moved, renamed or otherwise changed. Images may be imported again or already imported images may become unreadable.

Permissions

Changing the permissions of a directory within DropBox may result in duplicate imports as a newly readable directory appears identical to a new directory. If directories need to be modified it is recommended that the DropBox system is stopped and then restarted around any changes, as below.

As the **omero-server system user**, run

```
$ omero admin ice server disable DropBox
$ omero admin ice server stop DropBox
$ omero admin ice server disable MonitorServer
$ omero admin ice server stop MonitorServer

# make any directory changes

$ omero admin ice server enable MonitorServer
$ omero admin ice server enable DropBox
```

Note: Any new files copied into DropBox during this disabled period will not be detected and thus not imported.

Log files

The log files `var/log/FileServer.log`, `var/log/MonitorServer.log` and `var/log/DropBox.log` will indicate success or otherwise of start-up of the two components. Once running, `var/log/MonitorServer.log` will log file events seen within designated file areas and `var/log/DropBox.log` will log the progress of any file imports.

Unicode path and file names

If file or path names contain Unicode characters this can cause DropBox to fail. This can be remedied by the use of a `sitecustomize.py` or `usercustomize.py` file containing the following:

```
import sys
reload(sys)
sys.setdefaultencoding('utf-8')
```

For more details on using customization files in Python see: [site — Site-specific configuration hook](#). For more discussion on this issue within OMERO see the forum post: [Dropbox halts on certain unicode characters](#).

Note: If a customization file is used and the OMERO server is upgraded please ensure the file is still available to DropBox after the upgrade.

Advanced use

OMERO.dropbox can be configured in several ways through `etc/grid/templates.xml`. In its default configuration, as detailed above, it monitors the subdirectory DropBox of the OMERO data directory for all users.

A number of the properties in `templates.xml` accept a semi-colon separated list of values. This extended configuration allows a site to watch multiple directories, and configure each for a different user, a different type of file, etc. Any value missing from the configuration (e.g. `value="1;2"`) will be replaced by the default value.

One example alternative configuration would be to watch specific directories for specific users.

Note: Temporarily, the “importUsers” parameter is disabled, because of a bug. You can still configure the DropBox in a way which gives all the users the same Advanced configs. To achieve this, do not specify the “importUsers” parameter and always just use the “amy” or just the “zak” part of the other parameters or concatenate the “zak” parameters with “amy” parameters in the examples below.

In the example below two directories are monitored, one for user amy and one for zak:

```
<property name="omero.fs.importUsers" value="amy;zak"/>
<property name="omero.fs.watchDir" value="/home/amy/myData;/home/zak/work/data"/>
```

The remaining properties have been left at their default values for both users.

To limit DropBox to import only files belonging to specific image types the following property can be set,

```
<property name="omero.fs.readers" value="/home/amy/my_readers.txt;"/>
```

Here only the image types listed in `my_readers.txt` will be imported for the user amy while the system-wide `readers.txt` will be used for zak.

For a full description of the properties see below.

Properties

Each property takes the form of a single item or a semi-colon separated list of items. Where the item is a list, values within that list should be comma separated.

- importUsers (temporarily disabled)

The importUsers is either `default` or a list of OMERO user names. In the case of the value being `default`, the same configuration is applied to all users and each subsequent configuration setting should be a single value. In the case of this value being a list of users, each subsequent value should be a list of the same length as the number of users. The default value is `default`.

```
<property name="omero.fs.importUsers" value="default"/>
```

- watchDir

The absolute directory path of interest for each user. The default is empty.

```
<property name="omero.fs.watchDir" value=""/>
```

- eventTypes

For automatic import Creation and Modification events are monitored. It is also possible to monitor Deletion events though these are not used by DropBox. The default is `Creation,Modification`.

```
<property name="omero.fs.eventTypes" value="Creation,Modification"/>
```

- pathMode

By default existing and newly created subdirectories are monitored. It is possible to restrict monitoring to a single directory (“Flat”), only existing subdirectories (“Recurse”), or all subdirectories (“Follow”). For DropBox to function correctly the mode should be Follow. The default is Follow.

```
<property name="omero.fs.pathMode" value="Follow"/>
```

- whitelist

A list of file extensions of interest. An empty list implies all file extensions are monitored. The default is an empty list.

```
<property name="omero.fs.whitelist" value=""/>
```

- blacklist

A list of subdirectories to ignore. Not currently supported.

```
<property name="omero.fs.blacklist" value=""/>
```

- timeout

This timeout in seconds is used by one-shot monitors. This property is not used by DropBox.

```
<property name="omero.fs.timeout" value="0.0"/>
```

- blockSize

The number of events that should be propagated to DropBox in one go. Zero implies all events possible. The default is zero.

```
<property name="omero.fs.blockSize" value="0"/>
```

- ignoreSysFiles

If this is True events concerning system files, such as filenames beginning with a dot or default new folder names, are ignored. The exact events ignored will be OS-dependent. The default is True.

```
<property name="omero.fs.ignoreSysFiles" value="True"/>
```

- ignoreDirEvents

If this is True then the creation and modification of subdirectories is not reported to DropBox. The default is True.

```
<property name="omero.fs.ignoreDirEvents" value="True"/>
```

- dirImportWait

The time in seconds that DropBox should wait after being notified of a file before starting an import on that file. This allows for companion files or filesets to be copied. If a new file is added to a fileset during this wait period DropBox begins waiting again. The default is 60 seconds.

```
<property name="omero.fs.dirImportWait" value="60"/>
```

- fileBatch

The number of files that can be copied in before processing the batch. In cases where there are large numbers of files in a typical file set it may be more efficient to set this value higher. The default is 10.

```
<property name="omero.fs.fileBatch" value="10"/>
```

- throttleImport

The time in seconds that DropBox should wait after initiating an import before initiating a second import. If imports are started too close together connection issues can arise. The default is 10 seconds.

```
<property name="omero.fs.throttleImport" value="10"/>
```

- readers

A file of readers. If this is a valid file then it is used to filter those events that are of interest. Only files corresponding to a reader in the file will be imported. The default is empty.

```
<property name="omero.fs.readers" value=""/>
```

- importArgs

A string of extra arguments supplied to the importer. This could include, for example, an email address to report failed imports to: `--report --email test@example.com`. The default is empty. For details on available extra arguments see *Import images*.

```
<property name="omero.fs.importArgs" value=""/>
```

Example

Here is a full example of a configuration for two users:

```
<property name="omero.fs.importUsers" value="amy;zak"/>
<property name="omero.fs.watchDir" value="/home/amy/myData;/home/zak/work/data"/>
<property name="omero.fs.eventTypes" value="Creation,Modification;Creation,
↳Modification"/>
<property name="omero.fs.pathMode" value="Follow;Follow"/>
<property name="omero.fs.whitelist" value=";"/>
<property name="omero.fs.blacklist" value=";"/>
<property name="omero.fs.timeout" value="0.0;0.0"/>
<property name="omero.fs.blockSize" value="0;0"/>
<property name="omero.fs.ignoreSysFiles" value="True;True"/>
<property name="omero.fs.ignoreDirEvents" value="True;True"/>
<property name="omero.fs.dirImportWait" value="60;60"/>
<property name="omero.fs.fileBatch" value="10;10"/>
<property name="omero.fs.throttleImport" value="10;10"/>
<property name="omero.fs.readers" value="/home/amy/my_readers.txt;"/>
<property name="omero.fs.importArgs" value="-T \"regex:^(.*(?:<Container1>.*?))\";--
↳report --email zak@example.com"/>
```

See also:

Import images

Import targets

2.7.2 In-place import

In-place import allows files which are accessible from the OMERO.server's filesystem to be imported into OMERO without the need to upload them over an OMERO login session. This requires users to have shell (SSH, etc.) access to the server machine, and so there are a number of *limitations* to this implementation. Development of this feature is on-going, with improvements planned to enable a more user-friendly experience. This CLI-based stop-gap is being made available at this stage because for some users, in-place import is essential for their use of OMERO.

This feature is designed to allow imaging facilities to import large datasets into OMERO while keeping them safely stored on the file system in a secure location that is *read-only* for users. Leaving the data in a user's file system **is very dangerous** as they may forget they need to keep it or move to a different institution. **Under no circumstances should in-place import be used with temporary storage.**

Warning: The instructions below should help you get started but **it is critical that you understand the implications of using this feature.** Please do not just copy commands and hope for the best.

Responsibilities

As a data management platform, OMERO assumes that it is in control of your data in order to help prevent data loss. It assumes that data was copied into the server and only a server administrator or authorized OMERO user would have the rights to do anything destructive to that data.

With in-place import, the data either resides completely outside of OMERO or is shared with other users. This means that the critical, possibly sole, copy of your data must be protected outside of OMERO. **This is your responsibility for the lifetime of the data.**

Limitations

In-place import is only available on the OMERO server system itself. In other words, using SSH or similar, you will need to shell into the server and run the command-line importer directly. If you are uncomfortable with this, you should let someone else handle in-place importing.

Someone wanting to perform an in-place import **MUST** have:

- a regular OMERO account
- an OS (Operating System)-account with access to `bin/omero`
- read access to the location of the data
- **write** access to the *ManagedRepository* or one of its subdirectories

The above means that it may be useful to create a single OS account (e.g. "import_user") which multiple users can log into, and then use their own OMERO accounts to import data. Alternatively, each OMERO user can be given an OS account with access rights to the data storage as well as the managed repository.

Also, there is still some data duplication when *pyramids* are generated. We are hoping to find a workaround for this in the future.

For soft linking with `--transfer=ln_s` it has been noticed that some plate imports run rather more slowly than usual. Other operations may also be affected. In determining if or how to use in-place import at your high-content screening facility, we thus recommend time profiling with representative data, and alerting us to any significant disappointments.

Warning: Do not use soft links when pointing to data inside the ManagedRepository. If the originals are deleted, the data will be lost.

Safety tips

Whether you chose to use the hard- or soft-linking option below, you should take steps to secure files which are in-place imported to OMERO. The best option is making them **read-only** for both the OMERO user and also for the owner of the data. This means the server cannot accidentally modify the files (e.g. if a client mixes up the file IDs and tries to write to the wrong location) and that the files cannot be removed from the system while OMERO is still using them. Files may not be renamed or otherwise altered such that the OMERO server user cannot find them at the expected location.

If possible, **all the files should be added to your regular backup process**. If the files for imported images are later removed or corrupted, the result will probably be that while the images remain in their projects or screens with their annotations and basic metadata, they simply cannot be successfully viewed. However, this behavior is **not guaranteed**, so do *not* assume that the resulting problems will not extend further. Once the problem is noticed, replacing the original image files from backups, in the same place with the same name, is likely but **not guaranteed** to fully restore the images and their associated data in OMERO.

Additional setup requirements

In-place import requires additional user and group setup. As no one should be allowed to log into the account used to install the server, to permit in-place imports you need to create a different user account, allowing someone to log into the server but not accidentally delete any files. Therefore, you should set up an ‘in-place’ user and an ‘in-place’ group and configure a subset of directories under *ManagedRepository* to let members of that group write to them. Important criteria include:

- In-place users can write to directories that are newly created for import so that they may link out to the original file locations.
- In-place users cannot write to directories not required for their imports.
- In-place users cannot corrupt or delete each other’s imports.
- OMERO.server can read and delete all the imported files.

One may achieve the above with careful setting of sticky bits and choice of umasks or use of ACLs. The best approach depends on the background of your system administrators and the capabilities of the underlying filesystems. The example below details how this was done for one of our test servers in Dundee which runs with the default setting for `omero.fs.repo.path`:

```
### STATUS BEFORE

[sysadmin@ome-server omero_system_user]$ umask
0002

[sysadmin@ome-server omero_system_user]$ ls -ltrd ManagedRepository/
drwxrwxr-x 8 omero_system_user omero_system_user 4096 Apr 24 10:13 ManagedRepository/

[sysadmin@ome-server omero_system_user]$ grep inplace /etc/passwd /etc/group
/etc/passwd:inplace_user:x:501:501::/home/inplace_user:/bin/bash
/etc/group:omero_system_user:x:500:inplace_user
/etc/group:inplace_user:x:501:
```

(continues on next page)

(continued from previous page)

```

[sysadmin@ome-server omero_system_user]$ grep omero_system_user /etc/passwd /etc/group
/etc/passwd:omero_system_user:x:500:500::/home/omero_system_user:/bin/bash
/etc/group:omero_system_user:x:500:inplace_user

[sysadmin@ome-server omero_system_user]$ sudo -u inplace_user -i
[inplace_user@ome-server ~]$ umask
0002

### SCRIPT
chgrp inplace_user /repositories/binary-repository/ManagedRepository
chmod g+rws /repositories/binary-repository/ManagedRepository

chmod g+rws /repositories/binary-repository/ManagedRepository/*
chmod g+rws /repositories/binary-repository/ManagedRepository/*/*
chmod g+rws /repositories/binary-repository/ManagedRepository/*/*/*

chgrp inplace_user /repositories/binary-repository/ManagedRepository/*
chgrp inplace_user /repositories/binary-repository/ManagedRepository/*/*
chgrp inplace_user /repositories/binary-repository/ManagedRepository/*/*/*

# With the above, newly created directories should be in the inplace group
# As long as the file is readable by omero_system_user, then it should work fine!

### AFTER SCRIPT

[root@ome-server omero_system_user]# ls -ltrad ManagedRepository/
drwxrwsr-x 8 omero_system_user inplace_user 4096 Apr 24 10:13 ManagedRepository/

### TEST

# with default umask this likely has to do
[inplace_user@ome-server ~]$ cd /repositories/binary-repository/ManagedRepository/
[inplace_user@ome-server ManagedRepository]$ mkdir inplace.test
[inplace_user@ome-server ManagedRepository]$ ls -ltrad inplace.test/
drwxrwsr-x 2 inplace_user inplace_user 4096 Apr 30 11:35 inplace.test/

[omero_system_user@ome-server omero_system_user]$ cd /repositories/binary-repository/
↪ManagedRepository/
[omero_system_user@ome-server ManagedRepository]$ rmdir inplace.test/
[omero_system_user@ome-server ManagedRepository]$

```

If you are controlling OMERO.server with systemd you should add `UMask=0002` to the Service section of your systemd service file.

Getting started

The command-line import client has a help menu which explains the available options:

```
$ omero import --advanced-help
```

ADVANCED OPTIONS:

These options are not intended for general use. Make sure you have read the documentation regarding them. They may change in future releases.

In-place imports:

<code>--transfer=ARG</code>	File transfer method
General options:	
<code>upload</code>	# Default
<code>upload_rm</code>	# Caution! File upload followed by source deletion.
<code>some.class.Name</code>	# Use a class on the CLASSPATH.

Server-side options:

<code>ln</code>	# Use hard-link.
<code>ln_s</code>	# Use soft-link.
<code>ln_rm</code>	# Caution! Hard-link followed by source deletion.
<code>cp</code>	# Use local copy command.
<code>cp_rm</code>	# Caution! Copy followed by source deletion.

```
e.g. $ omero import --transfer=ln_s foo.tiff
$ ./importer-cli --transfer=ln bar.tiff
$ CLASSPATH=mycode.jar ./importer-cli --transfer=com.example.MyTransfer baz.tiff
```

Background imports:

<code>--keep-alive=SECS</code>	Frequency in seconds for pinging the server.
<code>--auto-close</code>	Close completed imports immediately.
<code>--minutes-wait=ARG</code>	Choose how long the importer will wait on server-
↳ side processing.	
	ARG > 0 implies the number of minutes to wait.
	ARG = 0 exits immediately. Use a *-completed option.
↳ to clean up.	
	ARG < 0 waits indefinitely. This is the default.
<code>--close-completed</code>	Close completed imports.
<code>--wait-completed</code>	Wait for all background imports to complete.

```
e.g. $ omero import -- --minutes-wait=0 file1.tiff file2.tiff file3.tiff
```

(continues on next page)

(continued from previous page)

```

$ ./importer-cli --minutes-wait=0 some_directory/
$ ./importer-cli --wait-completed # Waits on all 3 imports.

File exclusion:
-----

--exclude=filename           Exclude files based on filename.

--exclude=clientpath         Exclude files based on the original path.

e.g. $ omero import --exclude=filename foo.tiff # First-time imports
      $ omero import --exclude=filename foo.tiff # Second-time skips

Import speed:
-----

--checksum-algorithm=ARG      Choose a possibly faster algorithm for detecting
↪ file corruption,
                               e.g. Adler-32 (fast), CRC-32 (fast), File-Size-64
↪ (fast),
                               MD5-128, Murmur3-32, Murmur3-128,
                               SHA1-160 (slow, default)

e.g. $ omero import --checksum-algorithm=CRC-32 foo.tiff
      $ ./importer-cli --checksum-algorithm=Murmur3-128 bar.tiff

--no-stats-info              Disable calculation of minima and maxima when as part
↪ of the Bio-Formats reader metadata

e.g. $ omero import -- --no-stats-info foo.tiff
      $ ./importer-cli --no-stats-info bar.tiff

--no-thumbnails              Do not perform thumbnailing after import

e.g. $ omero import -- --no-thumbnails foo.tiff
      $ ./importer-cli --no-thumbnails bar.tiff

--no-upgrade-check            Disable upgrade check for each import
e.g. $ omero import -- --no-upgrade-check foo.tiff
      $ ./importer-cli --no-upgrade-check bar.tiff

Feedback:
-----

--qa-baseurl=ARG             Specify the base URL for reporting feedback
e.g. $ omero import broken_image.tif -- --email EMAIL --report --upload --logs --qa-
↪ baseurl=http://qa.example.com
      $ ./importer-cli broken_image.tif --email EMAIL --report --upload --logs --qa-
↪ baseurl=http://qa.openmicroscopy.org.uk/qa

```

(continues on next page)

(continued from previous page)

Report bugs at <https://www.openmicroscopy.org/forums>

The option for performing an in-place transfer is `--transfer`. A new extension point, file transfers allow a choice of which mechanism is used to get a file into OMERO.

```
$ omero import --transfer=ln_s my_file.dv
```

```
Using session bba923bb-cf0c-4cf0-80c5-a309be523ad8 (root@localhost:4064). Idle timeout:
↳10.0 min. Current group: system
...[    main] INFO          ome.formats.importer.ImportConfig - OMERO Version: 5.0.0-
↳rc1-DEV-ice35
...[    main] INFO          ome.formats.importer.ImportConfig - Bioformats version: 5.
↳0.0-rc1-DEV-ice35 revision: 101008f date: 31 January 2014
...[    main] INFO    formats.importer.cli.CommandLineImporter - Setting transfer to ln_
↳s
...[    main] INFO    formats.importer.cli.CommandLineImporter - Log levels -- Bio-
↳Formats: ERROR OMERO.importer: INFO
...[    main] INFO          ome.formats.importer.ImportCandidates - Depth: 4 Metadata
↳Level: MINIMUM
...[    main] INFO          ome.formats.importer.ImportCandidates - 1 file(s) parsed into
↳1 group(s) with 1 call(s) to setId in 98ms. (100ms total) [0 unknowns]
...[    main] INFO          ome.formats.OMEROMetadataStoreClient - Attempting initial SSL
↳connection to localhost:4064
...[    main] INFO          ome.formats.OMEROMetadataStoreClient - Insecure connection
↳requested, falling back
...[    main] INFO          ome.formats.OMEROMetadataStoreClient - Server: 5.0.0
...[    main] INFO          ome.formats.OMEROMetadataStoreClient - Client: 5.0.0-rc1-DEV-
↳ice35
...[    main] INFO          ome.formats.OMEROMetadataStoreClient - Java Version: 1.7.0_51
...[    main] INFO          ome.formats.OMEROMetadataStoreClient - OS Name: Linux
...[    main] INFO          ome.formats.OMEROMetadataStoreClient - OS Arch: amd64
...[    main] INFO          ome.formats.OMEROMetadataStoreClient - OS Version: 3.8.0-27-
↳generic
...[    main] INFO    ormats.importer.cli.LoggingImportMonitor - FILESET_UPLOAD_
↳PREPARATION
...[    main] INFO          ome.formats.importer.ImportConfig - OMERO Version: 5.0.0-
↳rc1-DEV-ice35
...[    main] INFO          ome.formats.importer.ImportConfig - Bioformats version: 5.
↳0.0-rc1-DEV-ice35 revision: 101008f date: 31 January 2014
...[    main] INFO    ormats.importer.cli.LoggingImportMonitor - FILESET_UPLOAD_START
...[    main] INFO    s.importer.transfers.SymlinkFileTransfer - Transferring /tmp/a.
↳fake...
...[    main] INFO    ormats.importer.cli.LoggingImportMonitor - FILE_UPLOAD_STARTED: /
↳tmp/a.fake
...[    main] INFO    ormats.importer.cli.LoggingImportMonitor - FILE_UPLOAD_COMPLETE: /
↳tmp/a.fake
...[    main] INFO    ormats.importer.cli.LoggingImportMonitor - FILESET_UPLOAD_END
...[l.Client-1] INFO    ormats.importer.cli.LoggingImportMonitor - METADATA_IMPORTED
↳Step: 1 of 5 Logfile: 24605
...[l.Client-0] INFO    ormats.importer.cli.LoggingImportMonitor - PIXELDATA_PROCESSED
↳Step: 2 of 5 Logfile: 24605
...[l.Client-1] INFO    ormats.importer.cli.LoggingImportMonitor - THUMBNAILS_GENERATED
```

(continues on next page)

(continued from previous page)

```

→ Step: 3 of 5 Logfile: 24605
...[l.Client-0] INFO   ormats.importer.cli.LoggingImportMonitor - METADATA_PROCESSED_
→ Step: 4 of 5 Logfile: 24605
...[l.Client-1] INFO   ormats.importer.cli.LoggingImportMonitor - OBJECTS_RETURNED Step:_
→ 5 of 5 Logfile: 24605
...[l.Client-0] INFO   ormats.importer.cli.LoggingImportMonitor - IMPORT_DONE Imported_
→ file: /tmp/a.fake
Imported pixels:
5001
Other imported objects:
Fileset:4102
Image:5001
...[l.Client-0] INFO   ome.formats.importer.cli.ErrorHandler - Number of errors: 0

```

The only visible difference here is the line:

```
...formats.importer.cli.CommandLineImporter - Setting transfer to ln_s
```

Rather than uploading via the OMERO API, the command-line importer makes a call to the system *ln* command.

Transfer options

Previously, OMERO only offered the option of uploading via the API. Files were written in blocks via the *RawFileStore* interface. With in-place import, several options are provided out of the box as well as the ability to use your own.

“ln_s” - soft-linking

The most flexible option is soft-linking. For each file, it executes *ln -s source target* on the local file system. This works across file system boundaries and leaves a clear record of what file was imported:

```

/OMERO/ManagedRepository/root_0/2014-01/24/10-11-14.947$ ls -ltra
total 8
lrwxrwxrwx 1 omero omero   11 Jan 24 10:11 my-file.dv -> /home/demo/my-file.dv

```

Here you can see in the imported file set, a soft-link which belongs to the *omero* user, but which points to a file in the */home/demo* directory.

Deleting the imported images in OMERO will delete the **soft link** but **not** the original file under */home*. This could come as a surprise to users, since the deletion will effectively free no space.

Warning: The deletion of the original files under */home* (or equivalent) **will lead to a complete loss of the data since no copy is held in OMERO**. Therefore, this method should only be used in conjunction with a properly managed and backed-up data repository. If the files are corrupted or deleted, there is no way to use OMERO to retrieve them.

“ln” - hard-linking

The safest option is hard-linking, though it cannot be used across file systems. For each file, it executes *ln source target*. Attempting to hard link across file system boundaries will lead to an error:

```
...[      main] INFO          ome.formats.importer.ImportConfig - OMERO Version: 5.0.0-rc1-DEV-ice35
...[      main] INFO          ome.formats.importer.ImportConfig - Bioformats version: 5.0.0-rc1-DEV-ice35 revision: 101008f date: 31 January 2014
...[      main] INFO    formats.importer.cli.CommandLineImporter - Setting transfer to ln
...[      main] INFO    formats.importer.cli.CommandLineImporter - Log levels -- Bio-Formats: ERROR OMERO.importer: INFO
...[      main] INFO          ome.formats.importer.ImportCandidates - Depth: 4 Metadata Level: MINIMUM
...[      main] INFO          ome.formats.importer.ImportCandidates - 1 file(s) parsed into 1 group(s) with 1 call(s) to setId in 96ms. (98ms total) [0 unknowns]
...[      main] INFO          ome.formats.OMEROMetadataStoreClient - Attempting initial SSL connection to localhost:4064
...[      main] INFO          ome.formats.OMEROMetadataStoreClient - Insecure connection requested, falling back
...[      main] INFO          ome.formats.OMEROMetadataStoreClient - Server: 5.0.0
...[      main] INFO          ome.formats.OMEROMetadataStoreClient - Client: 5.0.0-rc1-DEV-ice35
...[      main] INFO          ome.formats.OMEROMetadataStoreClient - Java Version: 1.7.0_51
...[      main] INFO          ome.formats.OMEROMetadataStoreClient - OS Name: Linux
...[      main] INFO          ome.formats.OMEROMetadataStoreClient - OS Arch: amd64
...[      main] INFO          ome.formats.OMEROMetadataStoreClient - OS Version: 3.8.0-27-generic
...[      main] INFO    ormats.importer.cli.LoggingImportMonitor - FILESET_UPLOAD_PREPARATION
...[      main] INFO          ome.formats.importer.ImportConfig - OMERO Version: 5.0.0-rc1-DEV-ice35
...[      main] INFO          ome.formats.importer.ImportConfig - Bioformats version: 5.0.0-rc1-DEV-ice35 revision: 101008f date: 31 January 2014
...[      main] INFO    ormats.importer.cli.LoggingImportMonitor - FILESET_UPLOAD_START
...[      main] INFO    .importer.transfers.HardlinkFileTransfer - Transferring /tmp/a.fake...
...[      main] INFO    ormats.importer.cli.LoggingImportMonitor - FILE_UPLOAD_STARTED: /tmp/a.fake
...[      main] ERROR    .importer.transfers.HardlinkFileTransfer - transfer process returned 1
...[      main] ERROR    .importer.transfers.HardlinkFileTransfer - error in closing raw file store
omero.ResourceError: null
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method) ~[na:1.7.0_51]
    at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:57) ~[na:1.7.0_51]
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45) ~[na:1.7.0_51]
    at java.lang.reflect.Constructor.newInstance(Constructor.java:526) ~[na:1.7.0_51]
    at java.lang.Class.newInstance(Class.java:374) ~[na:1.7.0_51]
    at IceInternal.BasicStream.createUserException(BasicStream.java:2615) ~[ice.jar:na]
```

(continues on next page)

(continued from previous page)

```

    at IceInternal.BasicStream.access$300(BasicStream.java:12) ~[ice.jar:na]
    at IceInternal.BasicStream$EncapsDecoder10.throwException(BasicStream.java:3099) ~
    ↪ [ice.jar:na]
    at IceInternal.BasicStream.throwException(BasicStream.java:2077) ~[ice.jar:na]
    at IceInternal.Outgoing.throwUserException(Outgoing.java:538) ~[ice.jar:na]
    at omero.api._RawFileStoreDelM.close(_RawFileStoreDelM.java:466) ~[omero-blitz.
    ↪ jar:na]
    at omero.api.RawFileStorePrxHelper.close(RawFileStorePrxHelper.java:1739) ~[omero-
    ↪ blitz.jar:na]
    at omero.api.RawFileStorePrxHelper.close(RawFileStorePrxHelper.java:1701) ~[omero-
    ↪ blitz.jar:na]
    at ome.formats.importer.transfers.AbstractFileTransfer.
    ↪ cleanupUpload(AbstractFileTransfer.java:123) ~[omero-blitz.jar:na]
    at ome.formats.importer.transfers.AbstractExecFileTransfer.
    ↪ transfer(AbstractExecFileTransfer.java:63) [omero-blitz.jar:na]
    at ome.formats.importer.ImportLibrary.uploadFile(ImportLibrary.java:410) [omero-
    ↪ blitz.jar:na]
    at ome.formats.importer.ImportLibrary.importImage(ImportLibrary.java:465) [omero-
    ↪ blitz.jar:na]
    at ome.formats.importer.ImportLibrary.importCandidates(ImportLibrary.java:274) ↪
    ↪ [omero-blitz.jar:na]
    at ome.formats.importer.cli.CommandLineImporter.start(CommandLineImporter.java:218) ↪
    ↪ [omero-blitz.jar:na]
    at ome.formats.importer.cli.CommandLineImporter.main(CommandLineImporter.java:658) ↪
    ↪ [omero-blitz.jar:na]
2014-01-31 12:59:20,338 3152      [      main] ERROR      ome.formats.importer.
    ↪ ImportLibrary - Error on import
java.lang.RuntimeException: transfer process returned 1
    at ome.formats.importer.transfers.AbstractExecFileTransfer.
    ↪ exec(AbstractExecFileTransfer.java:137) ~[omero-blitz.jar:na]
    at ome.formats.importer.transfers.AbstractExecFileTransfer.
    ↪ transfer(AbstractExecFileTransfer.java:57) ~[omero-blitz.jar:na]
    at ome.formats.importer.ImportLibrary.uploadFile(ImportLibrary.java:410) ~[omero-
    ↪ blitz.jar:na]
    at ome.formats.importer.ImportLibrary.importImage(ImportLibrary.java:465) ~[omero-
    ↪ blitz.jar:na]
    at ome.formats.importer.ImportLibrary.importCandidates(ImportLibrary.java:274) ~
    ↪ [omero-blitz.jar:na]
    at ome.formats.importer.cli.CommandLineImporter.start(CommandLineImporter.java:218) ↪
    ↪ [omero-blitz.jar:na]
    at ome.formats.importer.cli.CommandLineImporter.main(CommandLineImporter.java:658) ↪
    ↪ [omero-blitz.jar:na]
2014-01-31 12:59:20,338 3152      [      main] INFO      ome.formats.importer.
    ↪ ImportLibrary - Exiting on error

```

The safeness of this method comes from the fact that OMERO *also* has a pointer to the data. Deletion of the original file under `/home` would leave data in OMERO in place. Again, this could cause a surprise as the space would not be properly freed, but at least there cannot be an accidental loss.

Warning: The primary concern with this method is **modification** of files. If the original data is *written* by a user, unexpected results could follow in OMERO. See the *Safety tips* section above for ways around this.

If you are unclear about how hard-linking works, please see the [Hard link](#) article on Wikipedia.

The semantics of hard-linking have changed recently on Linux systems with the “protected hardlinks” feature, which is enabled by default and is in use on Ubuntu 22.04, RockyLinux/RHEL 9 and other contemporary systems. When you create a hard-link to a file, Linux now requires that you are either the *owner* of the file, or that you have *read-write permissions* to the file. Other Unix systems, and older Linux systems, allow a hard-link to be made if you have *search access* to the file (i.e. you have appropriate read and execute permissions on the directory path containing the file), but do not check the file permissions themselves. See the [kernel-hardening mailing list post](#) which describes the change in more detail. The implication for in-place import is that the user performing the import must own or have read-write permissions on the data files being imported in-place.

“ln_rm” - moving

Finally, the least favored option is `ln_rm`. It first performs a hard-link like `ln`, but once the import is complete it attempts to delete the original file. This is currently in testing as an option for DropBox but is unlikely to be of use to general users. Although this option is more limited than the `upload_rm` option below it will be much faster.

“upload_rm” - uploading and deleting

This option is not strictly an in-place option but is detailed here for convenience. It first performs a file upload like default import, but once the import is complete it attempts to delete the original files. It deletes the original files **if and only if** the import is successful.

“cp” and “cp_rm” variants

The `cp` and `cp_rm` commands provide the same functionality as `ln` and `ln_rm` but perform a copy rather than a link operation. The benefit of a copy is that it works over OS filesystem boundaries while still providing the integrity that `ln_s` cannot. The primary downside of a raw `cp` compared to `ln` is that there is data duplication. `cp_rm` being very similar to `ln_rm` usually works around this downside, except in the case of a failed import. Then the duplicated data will remain in OMERO and an explicit cleanup step will need to be taken.

Your own file transfer

If none of the above options work for you, it is also possible to write your own implementation of the `ome.formats.importer.transfers.FileTransfer` class, likely subclassing `ome.formats.importer.transfers.AbstractFileTransfer` or `ome.formats.importer.transfers.AbstractExecFileTransfer`. If you do so, please let us know how we might improve either the interface or the implementations that we provide.

Once your implementation has been compiled into a jar and placed in the `lib/clients` directory, you can invoke it using:

```
$ omero import --transfer=example.package.ClassName ...
```

Related advanced options

In addition to the `--transfer` option, a number of other advanced options have been added which may be useful for either tweaking import performance or dealing with complicated situations. Comments and suggestions are very welcome.

Checksums

If you think that calculating the checksums for your large files is consuming too much time, you might want to configure the checksum algorithm used. This can be done with the `--checksum_algorithm` property. Available options are printed with the `--advanced-help` option and include Adler-32, CRC-32, MD5-128, Murmur3-32, Murmur3-128, and the default SHA1-160.

DropBox

As described in the scenarios “*DropBox import (automatic delete)*” and “*In-place DropBox import (automatic delete)*”, *DropBox* can be configured to use any of the options described above. The configuration property to modify is *omero.fs.importArgs*:

```
$ omero config set -- omero.fs.importArgs "--transfer=upload_rm"
```

This will **move** files into OMERO rather than leaving a copy in the DropBox directory.

```
$ omero config set -- omero.fs.importArgs "--transfer=ln_rm"
```

This will also **move** files into OMERO rather than leaving a copy in the DropBox directory. For this to work, the two directories will need to be on the same file system. This option is much faster than *upload_rm*. Please read “*ln_rm*” - *moving* carefully to ensure you fully understand the implications of using this option.

For more information on OMERO.fs, please visit *Import under OMERO.fs*.

Warning: Use at your own risk!

See also:

Advanced import scenarios

OMERO.dropbox

Import under OMERO.fs

Import images

Import targets

Command Line Interface as an OMERO client

2.7.3 Advanced import scenarios

Increasingly users of OMERO are needing to go beyond the traditional “upload via a GUI”-style import model to more powerful methods.

There is a set of requirements for getting data into OMERO that is common to many institutions. Some of the requirements may be mutually exclusive.

- Users need to get data off microscopes quickly. This likely includes not waiting for import to complete. Users will often move data immediately, or even save remotely during acquisition.
- Users would like direct access to the binary repository file-system to read original files for analysis.
- Users would like to view and begin working with images as soon as possible after acquisition.

Below we explain which options are available to you, and why there is a trade-off between the above requirements.

Import overview

The “OMERO binary repository” (or repo) is the directory belonging to the OMERO user where files are imported:

- The *ManagedRepository* directory inside of the repo is where files go during import into OMERO. Each user receives a top-level directory inside of “ManagedRepository” which fills with timestamped directories as imports accrue.
- Depending on the permissions of this directory, users may or may not be able to see their imported files. Managing the permissions is the responsibility of the system administrator.

In “normal import”, files are copied to the OMERO binary repo via the API and so can work remotely or locally. In “*in-place import*”, files are “linked” into place.

Warning: In-place import is a new, powerful feature - it is critical that you read and understand the [documentation](#) before you consider using it.

Traditional import

Manual import (GUI)

This is the standard workflow and the one currently used at the University of Dundee. Users dump data to a shared file-system from the acquisition system, and then use the OMERO.insight client from the lab to import.

Advantages

- Users can validate that import worked.
- Failed imports can be repeated and/or reported to QA etc.
- Users do not have to wait for import to be scheduled.
- Import destination is known: Project/Dataset etc.

Disadvantages

- Imports can be slow due to the data transfer from file-system to OMERO via the client.
- Users must remember to delete data from the shared file-system to avoid data duplication.
- Users cannot access the OMERO binary repo directly and must download original data via clients for local analysis.

Manual import (CLI)

Another typical workflow is that rather than using the GUI, users perform the same procedure as under “Manual import” but with the *command-line (CLI) importer*.

Advantages

- With a CLI workflow, it may be easier for users to connect remotely to kick off an import and to leave it running in the background for a long period of time.

Disadvantages

All the same disadvantages apply as under “Manual import (GUI)”.

Cronjob import (manual delete)

For importing via cron, users still dump their data to a shared file-system from the acquisition system. They must have permissions to write to “their” directory which is mapped to a user in OMERO.

A cronjob starts a CLI import, possibly at night. The cronjob could be given admin rights in OMERO to perform an “Import As” for a particular user.

Disadvantages

- If a normal import is used, the cronjob would have to manually delete imported files from their original location to avoid duplication.
- Users cannot work with their data in OMERO until some time after acquisition.
- Failed imports are logged within the managed repository but not yet notified. Logs would probably need to be accessed via a sysadmin/cli. The cronjob could capture stdout and stderr and check for failures.

DropBox import (manual delete)

Similar to the cronjob scenario, DropBox importing requires that users drop their data in “their” directory which has special permissions for writing. The DropBox service monitors those directories for modifications and imports the files on a first-come-first-serve basis.

- *OMERO.dropbox*

Advantages

- Users should see their data in OMERO quickly.

Disadvantages

- There is a limitation on the rate of new files in monitored locations.
- There is also a limitation on which file systems can be used. A networked file share **cannot** be monitored by DropBox.
- Users must manually delete imported files from their DropBox directory to avoid duplication.
- Failed imports are logged within the managed repository but not yet notified. Logs would probably need to be accessed via a sysadmin or through the CLI and searched by the user and file name.

DropBox import (automatic delete)

One option is to have files removed from DropBox automatically after a successful import. This is achieved by performing an “upload” import from the DropBox directory to the ManagedRepository then deleting the data from DropBox **if and only if** the import was successful. For failed imports, files will remain in the DropBox directories until someone manually deletes them.

Advantages

- For all successful imports, files will be automatically removed from the DropBox directories thus reducing duplication.

In-place import

The following sections outline *in-place* based scenarios to help you judge if the functionality may be useful for you.

Common 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.

Common disadvantages

- Like the DropBox import scenario above, all in-place imports require the user to have access to the user-based directories under the ManagedRepository. See [limitations](#) for more details.
- Similarly, all the following scenarios carry the same burden of securing the data externally to OMERO. This is the primary difference between a normal import and an in-place import: **backing up OMERO is no longer sufficient to prevent data loss. The original location must also be secured!** This means that users must not move or alter data once imported.

In-place manual import (CLI)

The in-place version of a CLI manual import is quite similar to the normal CLI import, with the primary difference being that the data is not transferred from the shared file-system where the data is initially stored after acquisition, but instead is just “linked” into place.

Advantages

- Local filesystem in-place import is faster than traditional importing, due to the lack of a data transfer.

Disadvantages

- Requires proper security setup as explained above.

In-place Cronjob import

Assuming all the restrictions are met, the cronjob-based workflow above can carry out an in-place import by adding the in-place transfer flag. The advantages and disadvantages are as above.

In-place DropBox import (manual delete)

Just as with the in-place cronjob import, using in-place import for DropBox is as straight-forward as passing the in-place flag. The common advantages and disadvantages of in-place import apply.

In-place DropBox import (automatic delete)

An option that also exists in the in-place scenario is to have files removed from DropBox automatically after a successful import. This is achieved by first performing a “hardlink in-place import” from the DropBox directory to the ManagedRepository and then by deleting the data from DropBox **if and only if** the import was successful. For failed imports, files will remain in the DropBox directories until someone manually deletes them.

Advantages

- For all successful imports, files will be automatically removed from the DropBox directories.

Disadvantages

- This option is only available if the filesystem which DropBox watches is the same as the file system which the ManagedRepository lives on. This prevents the use of network file systems and similar remote shares.

Parallel import

Parallel import is a variant of manual CLI import for making large-scale imports considerably faster. It is *experimental* and may see extensive changes between patch versions. Use of this feature entails risk: if high thread counts are specified then the import client or OMERO server may function poorly. New uses of parallel import should be tested with a non-production server. Experience gained within OME and reported by users will help to make parallel import more friendly and safe.

`omero import --parallel-fileset` sets how many filesets are imported at the same time. `omero import --parallel-upload` sets how many files are uploaded at the same time. File upload occurs early in import and the fileset import threads share the same file upload threads among them so it typically makes sense to set the file upload thread count at least as high as the fileset import thread count. They both default to a value of 1.

These options can provide clear benefits if set even at lower numbers like 4. Do not assume that higher is always better: more concurrent threads means higher overhead and may severely exhaust resources on the server and the client. Issues with parallel import include:

- Import can fail when the same repository directory is being created to hold the files from different filesets. An effective workaround is to set the server's *Template path* such that the `%thread%` term precedes any subdirectories that may need to be created at import time.
- Import can fail when the same *import target* is created to contain multiple filesets. An effective workaround is to create the targets in advance of starting the imports.
- The server's connections to the database may become saturated, making the server unresponsive. Set the `omero.db.poolsize` property higher than the number of filesets that will be imported across all users at any one time.

See also:

In-place import

OMERO.dropbox

Import images

Import targets

2.8 Optimizing OMERO as a Data Repository

This section explains how to customize the appearance and functionality of OMERO clients to host images for groups or public viewing.

2.8.1 Publishing data using OMERO.web

The OMERO.web framework allows raw data to be published using built-in tools or supplied through web services to external web pages. Selected datasets can be made visible to a ‘public user’ using the standard OMERO permissions system, ensuring you always have control over how users can interact with your data.

There are several ways of publishing data using OMERO.web:

- using a URL to launch the web-based Image viewer, as described in [Launching OMERO.web viewer](#), which can be accompanied by a thumbnail. For more details of how to load the thumbnail, see [URLs from within OMERO.web](#)
- embedding the image viewport directly into other web pages, for more details see [Customizing the content of the embedded OMERO.web viewport](#)
- allowing public access to the OMERO.web data manager
- writing your own app to host your public data (see [Creating an app](#)) and then allowing public access to the chosen URL for that app

The sections below describe how you might use these features and how to set them up.

Configuring public user

The OMERO.web framework supports auto-login for a single username / password. This means that any public visitors to certain OMERO.web pages will be automatically logged in and will be able to access the data available to the defined ‘public user’.

To set this up on your OMERO.web installation:

- Create a group with read-only permissions (the name can be anything e.g. “public-data”). We recommend read-only permissions so that the public user will not be able to modify, delete or annotate data belonging to other members.
- Create a member of this group, noting the username and password (you will enter these below). Again, the First name, Last name, Username and Password can be anything you like.

Note: If you add this member to other groups, all data in these groups will also become publicly accessible for as long as this user remains in the group.

- Enable the `omero.web.public.enabled` property and set `omero.web.public.user` and `omero.web.public.password`:

```
$ omero config set omero.web.public.enabled True
$ omero config set omero.web.public.user '<username>'
$ omero config set omero.web.public.password '<password>'
```

- By default the public user is only allowed to perform GET requests. This means that the public user will not be able to Create, Edit or Delete data, as these require POST requests. If you want to allow these actions from the public user, you can change the `omero.web.public.get_only` property:

```
$ omero config set omero.web.public.get_only false
```

- Set the `omero.web.public.url_filter`. This filter is a regular expression that will allow only matching URLs to be accessed by the public user. If this is not set, no URLs will be publicly available.

You need to configure the `url_filter` to *allow* all URLs that are required for the pages you wish to be public but to *block* any URLs that you do not want public users to access.

Some examples are listed below:

- To allow all URLs from a single app, such as ‘webgateway’, use a filter for URLs that start with the app name. For example:

```
$ omero config set omero.web.public.url_filter '^/webgateway'
```

This filter permits all URLs needed for the full image viewer. If you wish to block webgateway URLs for downloading data, use:

```
$ omero config set omero.web.public.url_filter '^/webgateway/(?!archived_
↪files|download_as)'
```

- You may need to allow access to additional URLs for some apps. For example, the [OMERO.iviewer](#) also uses some webgateway and api URLs:

```
$ omero config set omero.web.public.url_filter '^/iviewer|webgateway|api'
```

- You can use the full webclient UI for public browsing of images. Attempts by public user to create, edit or delete data will fail silently with the default `omero.web.public.get_only` setting above. You may also choose to disable various dialogs for these actions such as launching scripts or OME-TIFF export, for example:

```
$ omero config set omero.web.public.url_filter '^/(webadmin/myphoto/|webclient/
↪(?!(script_ui|ome_tiff|figure_script))|webgateway/(?!(archived_files|download_
↪as))|iviewer|api)'
```

- Set the `omero.web.public.server_id` which the public user will be automatically connected to. Default: 1 (the first server in the `omero.web.server_list`):

```
$ omero config set omero.web.public.server_id 1
```

If you enable public access to the main webclient but still wish registered users to be able to log in, the login page can always be accessed using a link of the form https://your_host/webclient/login/.

Full example of hosting data for a publication

Putting the pieces of this puzzle together, the following describes the steps of a complete workflow for using OMERO to host public data associated with a publication. It is illustrated using an example publication from the Swedlow lab in Dundee, [Schleicher et al, 2017](#) with the data hosted at <https://omero.lifesci.dundee.ac.uk/pub/schleicher-et-al-2017>.

Ansible playbooks can be found describing how the production server in Dundee (“nightshade”) was configured in the [prod-playbooks](#) repository on GitHub.

Group setup

A group-per-publication allows the public user to be selectively added (or removed) from given publications to decide their visibility.

1. Create a dedicated read-only group to host the raw data underlying the publication (see *User/group management*).
2. Add all the authors of the paper to this new group.
3. Once you have configured OMERO.web to create a public user (see below), add the public user as a member of the newly created read-only group.

Configuring OMERO.web

If you wish to have an automatically logged-in public user while still giving your existing OMERO users an unchanged user experience (i.e. not automatically logging them in as the public user), a dedicated, *separate web server* for servicing the public workflows can be added and configured to point at your existing OMERO.server. This is the workflow adopted here by adding a public OMERO.web at <https://omero.lifesci.dundee.ac.uk>, without changing the existing internal OMERO.web.

1. Follow the steps in *Configuring public user* above on the chosen OMERO.web.
2. Also configure *the filter on the public user* on the chosen OMERO.web by setting `omero.web.public.url_filter` to allow 'webclient' so that the full webclient is visible for the public user, and thus the Data tree with Projects and Datasets is also browsable, as well as the Tags tab and the full image viewer.

Data migration

The data to be made public will need to be in the publication group to be considered “published”.

1. Move the original images into the dedicated group using OMERO.web or *OMERO.cli*. The CLI is best used where Images or Datasets are cross-linked to other Datasets or Projects in the original group. The command `omero chgrp Project:$ID --include Dataset,Image` cuts the cross-links in the original group and preserves the Project/Dataset/Image hierarchy prepared for the move by the author.
2. If you have used OMERO.figure to create your figures for publication, you can always find the original data by using the 'info' tab inside the OMERO.figure app, see the *OMERO.figure creation guide* (OMERO.figure supports a complete figure creation workflow, including exporting figures into image processing applications for final adjustments - see the *OMERO.figure User guide* for full details).
3. Having all the data belong to one user simplifies the UI experience for public users. If necessary, ownership of data can be transferred using the 'Chown' privilege (see *Administrators with restricted privileges* and *Changing ownership of objects*).

Data layout

Once the data is in the dedicated read-only group, it can be reorganized and renamed to reflect the publication e.g. Projects can be renamed according to the corresponding figure panels in the manuscript while the names of the Datasets could be retained corresponding to different treatment conditions represented in each figure panel. For example, Project `Schleicher_etal_figure7_c` contains images underlying the publication Figure panel 7(c). Some Projects underlie two publication figure panels, such as Project `Schleicher_etal_figure2_a_c` where representative images are shown in panel (a) and the corresponding quantification is shown in panel (c) of Figure 2. This makes clear which original images are underlying which figure panels in the publication.

Data can also be tagged with OMERO tags to enhance the browsing possibilities through these data for any user with basic knowledge of OMERO. For example, see [Tag:Schleicher_etal_figure1_a](#). The tags are highlighting the images displayed in the publication figures as images. The other, non-tagged images in the group are the ones used for analysis which produced the published numerical data.

Key-Value pairs can be used to add more detailed information about the study and publication. For example, go to [Schleicher_etal_figure1_a](#) and expand the 'Key-Value Pairs' section in the right-hand pane to display the content (see the [Annotate User guide](#) for information on using Key-Value pairs).

Configuring URLs

The URL of the first Project (corresponding to the first figure in the publication) can be used for a DOI and data landing page. For example, Project 'Schleicher_etal_figure1_a' <https://omero.lifesci.dundee.ac.uk/webclient/?show=project-27936> corresponds to <http://dx.doi.org/10.17867/10000109>.

Optionally, you can decide on a set pattern of URLs for this and future publications. For example, in Dundee we have established a pattern which supposes every new publication from our institution will be in a separate group, and this group will be directly navigable by the public user using the syntax: "server-address/pub/publication-identifier". This means for example, <https://omero.lifesci.dundee.ac.uk/pub/schleicher-et-al-2017> is the link where "omero.lifesci.dundee.ac.uk" is the server address, and "schleicher-et-al-2017" is the publication-identifier.

This makes use of redirects allowing <https://omero.lifesci.dundee.ac.uk/pub/schleicher-et-al-2017> to link to the correct group and Project in OMERO, just as the DOI above does. Redirects need to be set in the NGINX component of the OMERO.web installation dedicated to publication workflows. You can find our configuration for this example [here on GitHub](#):

```
location /pub/schleicher-et-al-2017 {
    return 307 /webclient/?show=project-27936;
}
```

2.8.2 OMERO.web UI customization

The OMERO.web offer a flexible user interface that can be customized. The sections below describe how to set up these features.

Note that depending on the deployment choice, OMERO.web will not activate configuration changes until Gunicorn is restarted using **omero web restart**.

Index page

This allows you to add a homepage at <your-omero-server>/index/. Visitors to your root url at <your-omero-server>/ will get redirected here instead of redirecting to <your-omero-server>/webclient/.

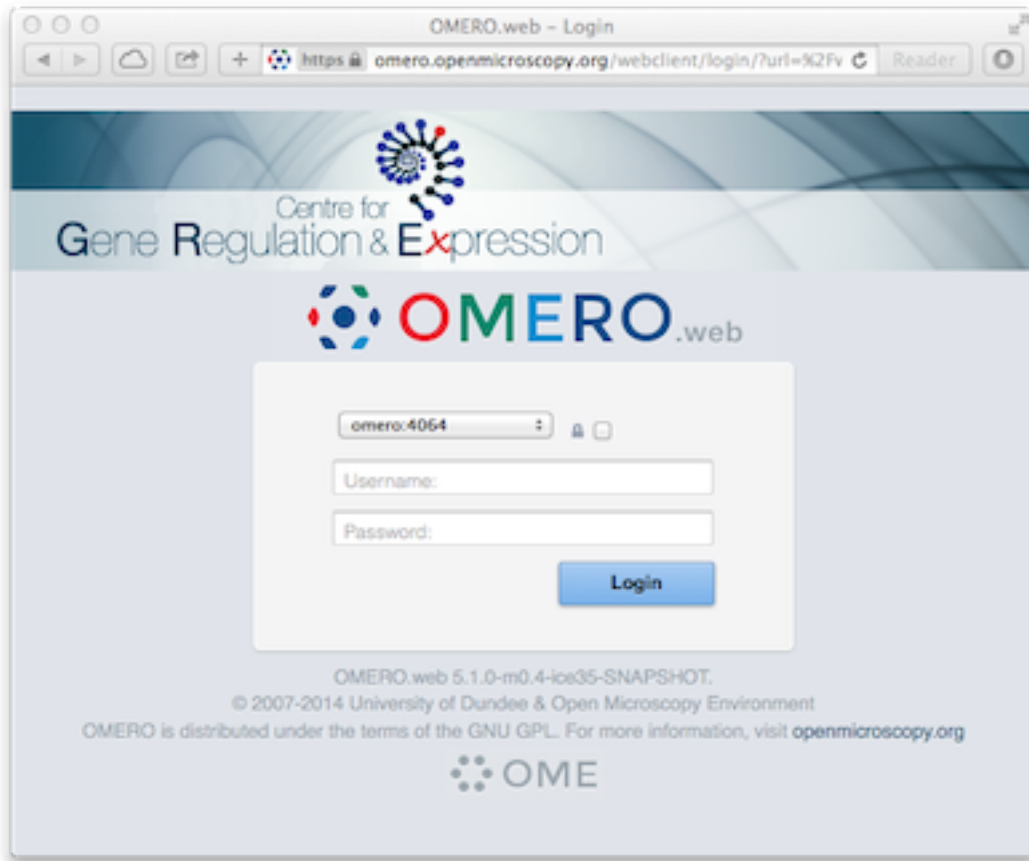
Create new custom template in /your/path/to/templates/mytemplate/index.html and add the following:

```
$ omero config append omero.web.template_dirs '/your/path/to/templates/'
$ omero config set omero.web.index_template 'mytemplate/index.html'
```

Login page logo

`omero.web.login_logo` allows you to customize the webclient login page with your own logo. Logo images should ideally be 150 pixels high or less and will appear above the OMERO logo. You will need to host the image somewhere else and link to it with:

```
$ omero config set omero.web.login_logo 'http://www.url/to/image.png'
```



Login redirection

`omero.web.login_redirect` property redirects to the given location after logging in to named pages. In the example below, a user who tries to visit the "webindex" URL (`/webclient/`) will be redirected after login to a URL defined by the `viewname` "load_template". The "args" are additional arguments to pass to Django's `reverse()` function and the "query_string" will be added to the URL:

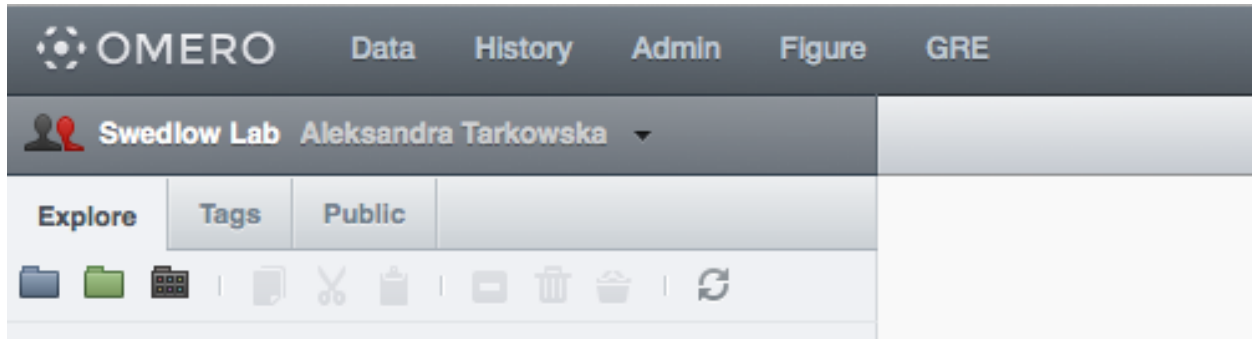
```
$ omero config set omero.web.login_redirect '{"redirect": ["webindex"], "viewname":  
→ "load_template", "args": ["userdata"], "query_string": "experimenter=-1"}'
```


Top links menu

`omero.web.ui.top_links` adds links to the top header:

```
$ omero config append omero.web.ui.top_links '["Figure", "figure_index", {"title": "Open_
↪Figure in new tab", "target": "_blank"}]'
```

```
$ omero config append omero.web.ui.top_links '["GRE", "http://lifesci.dundee.ac.uk/gre"]'
```



Open With option

`omero.web.open_with` adds items to the ‘Open with’ options. This allows users to open selected images or other data with another web app or URL. See [Linking from Webclient](#).

Include template in every page

An HTML template specified by `omero.web.base_include_template` will be included in every HTML page in OMERO.web. The template is inserted just before the `</body>` tag and can be used for adding a `<script>` such as Google analytics.

For example, create a file called `/your/path/to/templates/base_include.html` with:

```
<script>
  console.log("Hello World");
</script>
```

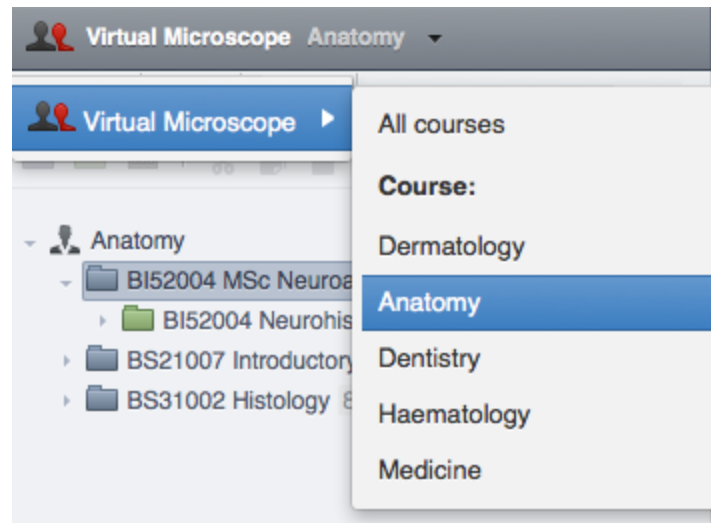
Set the following:

```
$ omero config append omero.web.template_dirs '"/your/path/to/templates/'
$ omero config set omero.web.base_include_template 'base_include.html'
```

Group and Users in dropdown menu

Customize the groups and users dropdown menu by changing the labels or hiding the entire list:

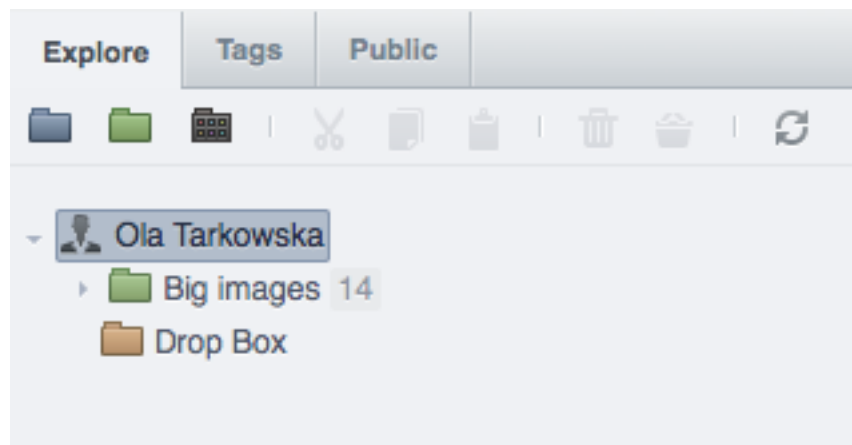
```
$ omero config set omero.client.ui.menu.dropdown.leaders.label "Owners"
$ omero config set omero.client.ui.menu.dropdown.leaders.enabled true
$ omero config set omero.client.ui.menu.dropdown.colleagues.label "Members"
$ omero config set omero.client.ui.menu.dropdown.colleagues.enabled true
$ omero config set omero.client.ui.menu.dropdown.everyone.label "All Members"
$ omero config set omero.client.ui.menu.dropdown.everyone.enabled false
```



Orphaned container

`omero.client.ui.tree.orphans.name` allows you to change the name of the “Orphaned images” container located in the client data manager tree:

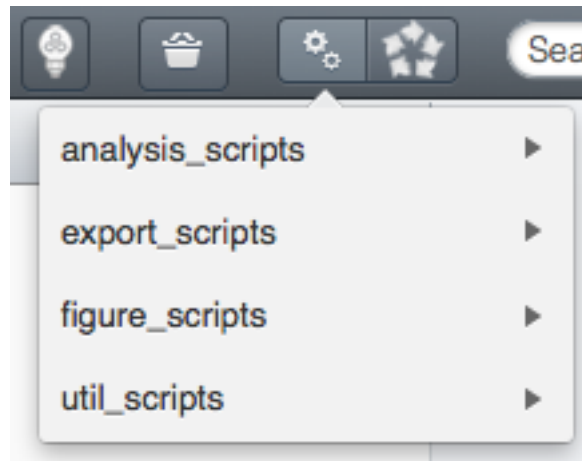
```
$ omero config set omero.client.ui.tree.orphans.name "Orphaned images"
```



Disabling scripts

`omero.client.scripts_to_ignore` hides the scripts that the clients should not display:

```
$ omero config append omero.client.scripts_to_ignore "/my_scripts/script.py"
```



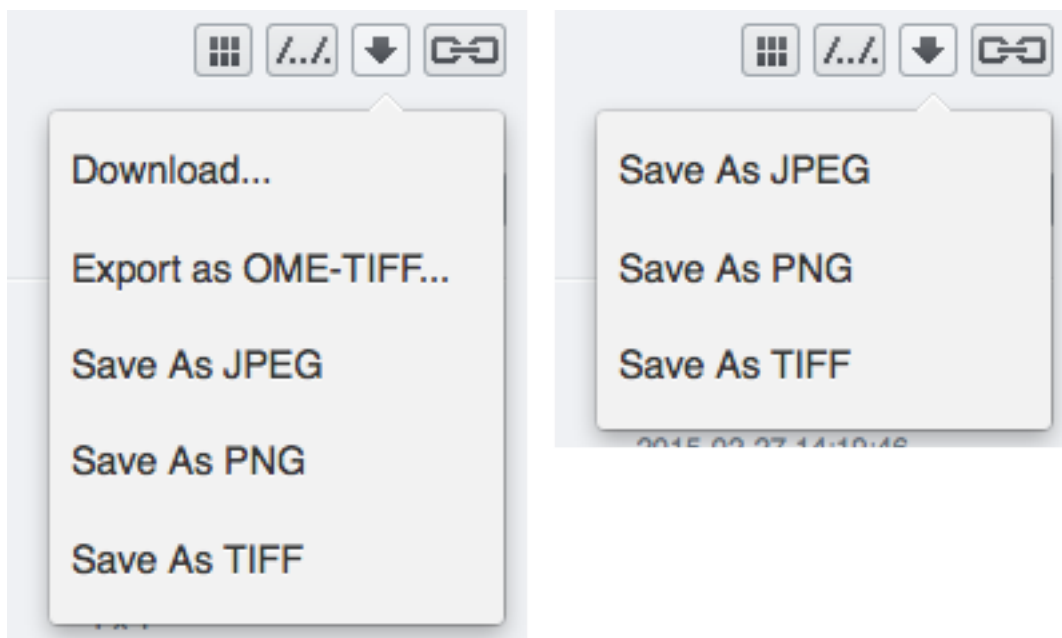
Download restrictions

`omero.policy.binary_access` determines whether users can access binary files from disk. Binary access includes all attempts to download a file from the UI:

```
$ omero config set -- omero.policy.binary_access +read,+write,+image
```

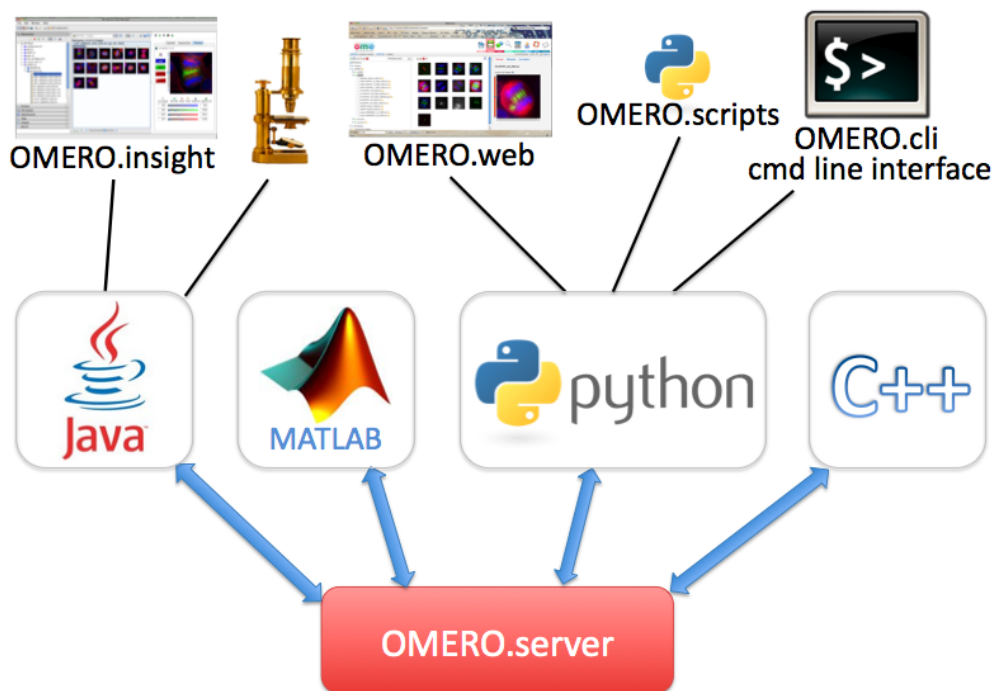
or on a specific group with ID 15:

```
$ omero group info 15
$ omero obj map-set ExperimenterGroup:15 config -- omero.policy.binary_access +read,
  ↳+write,+image
```



DEVELOPER DOCUMENTATION

The following documentation is for developers wishing to write OMERO client code or extend the OMERO server. Instructions on [downloading](#), installation and administering OMERO can be found under the *System Administrator Documentation* of the main site.



OMERO is an open source client/server system written in Java for visualizing, managing, and annotating microscope images and metadata. The *OMERO Application Programming Interface* allows clients to be written in *Java*, *Python*, *C++* or *MATLAB*. OMERO releases include a Java client OMERO.insight, a Python-based web client OMERO.web and the *Command Line Interface as an OMERO development tool*, which also uses Python. There is also an ImageJ plugin. OMERO can be extended by modifying these clients or by writing your own in any of the supported languages (see figure). OMERO also supports a *Scripting Service* which allows Python scripts to be run on the server and called from any of the other clients.

OMERO is designed, developed and released by the [Open Microscopy Environment](#), with contributions from [Glencoe Software, Inc.](#) OMERO is released under the [GNU General Public License \(GPL\)](#) with commercial licenses and customization available from [Glencoe Software, Inc.](#) You can read about how OMERO has developed since the project started in the [CHANGELOGS](#). We ask contributors to sign once the [Contributor License Agreement \(CLA\)](#). When signed, please do send the agreement to contributors at openmicroscopy.org.

For help with any aspect of OMERO, see details of our [forums](#) and [mailing lists](#).

3.1 Introduction to OMERO

3.1.1 What's new for OMERO 5.6 for developers

This release focuses on migrating all Python components to Python 3, and decoupling them into separate repositories with the benefit of permitting each to be released to PyPI independently:

- <https://github.com/ome/omero-py>
- <https://github.com/ome/omero-web>
- <https://github.com/ome/omero-dropbox>
- <https://github.com/ome/omero-marshall>

For details on migrating your own code to Python 3, see *Migration from OMERO 5.5 (Python 2) to OMERO 5.6 (Python 3)*.

You may also find the Sysadmins *Migration to Python 3* page useful.

Other changes which you need to be aware of:

- The *path* module is now named *omero_ext.path*.

For a full list of api changes, bug fixes and other improvements, see the *CHANGELOGS*.

3.1.2 Migration from OMERO 5.5 (Python 2) to OMERO 5.6 (Python 3)

This page serves as a collection of recommendations, developed as the OME team went through the migration to Python 3. This is not a complete guide but may serve as a useful starting point.

For more information, please see a dedicated Python 3 page like <http://python-future.org/>.

Futurize

Installing *future* from Python 3 is now required for all OMERO Python components. This library comes with the *futurize* tool which performs many of the basic transformations needed to migrate Python 2 code to Python 3:

```
futurize -0 your_file.py
```

Add *-w* to update the file in place.

print()

The most common transformation needed is adding parentheses around print statements since print is no longer a keyword.

dict.keys()

The return value from the *keys()* method of dictionaries is of type *dict_keys* and no longer has methods like *sort()*. Wrap with a call to *list* if you need the previous behavior: *list(my_dict.keys())*.

Strings

Changes to the handling of strings was our major hurdle in upgrading from Python 2 to Python 3. In Python 2, there is a separation between *str* and *unicode*. In Python 3, both of those are like *unicode* (but called *str*) and a new type was introduced: *bytes*. A good starting places to learn the difference is:

http://python-future.org/compatible_idioms.html?highlight=string#strings-and-bytes

The *future* library which enables support for Python 2 and Python 3 concurrently has its *own str* class. It is necessary to look at the imports for a module to know what *str* is being used.

Which *str* is it??

If nothing special is imported, *str* is the builtin *str* which in Python 2 is non-unicode and unicode in Python 3. String literals like “foo” are also of type *str*.

If *unicode_literal* is imported, then “foo” is the same as u”foo” and is *unicode* in Python 2 or just *str* in Python 3.

If *from builtins import str* is imported, then *str* is more like unicode and may fail existing calls to *isinstance()*.

isinstance(x, str)

Since *str* can change its type, this often will not do what you want. Using *past.builtins.basestring* is generally a good solution, e.g. *isinstance(x, basestring)*

str(some_variable)

If you are trying to turn a variable into a string, this may not do what you want since it might be creating a *unicode*.

This is especially problematic for passing strings to Ice methods, which are implemented in C++ and fail spectacularly if they receive non-string objects (like unicode).

future.utils.native_str maintains the previous semantics producing builtin *str* objects. Native str semantics are especially important when working with Ice, e.g.

```
ctx = {'omero.group': native_str(groupId)}
conn.getUpdateService().saveArray(pixels, ctx)
```

StringIO and open("file", "r")

StringIO and *open()* may need replacing with *BytesIO* and *open("file", "rb")* respectively. This depends on whether or not your code is expecting a stream to be binary.

Regexes

Regexes must start with *r""* for raw to prevent escapes from being misinterpreted (e.g. *d*).

Numerics

long no longer exists. Replace *omero.rtypes.wrap(long_value)* with *omero.rtypes.rlong(long_value)*.

Division with */* now produces a floating point. For example, *choice * int(percent) / 100* no longer produces an integer in Python 3. Use *//*.

3.1.3 Installing OMERO from source

Warning: Starting from OMERO 5.5, many components have been moved to their own repositories e.g. [OMERO.py](#), to modernize the application and allow more flexibility.

This page is currently under **review**.

Using the source code

The source code of each release of OMERO is available for download from the [Source code](#) section of the OMERO download page.

Note: At the moment, this source code bundle does not contain the version of Bio-Formats. To include this version information, you will need to manually copy the `ant/gitversion.xml` file included in the source code bundle of Bio-Formats for the same release under `components/bioformats/ant`.

Using the Git source repository

To use the Git source repository, you will need to install Git on your system. See the [Using Git](#) section of the Contributing documentation for more information on how to install and configure Git.

The main repository for OMERO is available from <https://github.com/ome/openmicroscopy>. Most OME development is currently happening on GitHub, therefore it is highly suggested that you become familiar with how it works, if not create an account for yourself.

Start by cloning the official repository:

```
git clone https://github.com/ome/openmicroscopy.git
```

Since the openmicroscopy repository now makes use of submodules, you first need to initialize all the submodules:

```
cd openmicroscopy
git submodule update --init
```


Alternatively, with version 1.6.5 of git and later, you can pass the `--recursive` option to git clone and initialize all submodules:

```
git clone --recursive https://github.com/ome/openmicroscopy.git
```

See also:

Using Git

Section of the contributing documentation explaining how to use Git for contributing to the source code.

Building OMERO

To install the dependencies required to run the OMERO.server on Linux or Mac OS X, take a look at the [OMERO.server installation](#) page where you will also find links to walk-throughs for specific platforms.

Some environment variables may need to be set up before building the server:

- If the system slice files cannot be found you must set `SLICEPATH` to point to the `slice` directory of the Ice installation.

Once all the dependencies and environment variables are set up, you can build the server using:

```
python build.py
```

or the clients using:

```
python build.py release-clients
```

See also:

Build System

Section of the developer documentation detailing the build system

3.1.4 Build System

Overview

The page goes into details about how the build system is configured.

Since 5.5, OMERO decouples many components and uses, for some components, an [Gradle](#)-based build. The two overarching repositories are <https://github.com/ome/omero-build> and <https://github.com/ome/omero-gradle-plugins>. See the README of each repository for more details. OMERO still uses an [Ant](#)-based build, for some components, with dependency management provided by [Ivy](#). [C++ code](#) is built using Cmake and Python uses the traditional distutils/setuptools tools.

Structure of the build

This is an (abbreviated) snapshot of the structure of the filesystem for OMERO:

```
OMERO_SOURCE_PREFIX
|
|-- build.xml ..... Top-level build file
|
|-- build.py ..... Python wrapper to handle OS-specific
↪ configuration
|
|-- omero.class ..... Self-contained Ant launcher
|
|--etc ..... Configuration folder
|   |-- grid ..... Deployment files folder
|   |-- ivysettings.xml ..... Main Ivy configuration file
|   |-- hibernate.properties
|   |-- build.properties
|   |-- logback.xml
|   |-- omero.properties
|   \-- profiles
|
|-- examples ..... User examples
|
\components
|
|--<component-name> ..... Each component has this same basic structure.
|   |-- build.xml ..... Build file
|   |-- ivy.xml ..... Jar dependencies
|   |-- test.xml ..... Test dependencies
|   |-- src ..... Source code
|   |-- resources ..... Other files of interest
|   |-- test ..... Test source code and test resources
|   \-- target ..... Build output (deleted on clean)
|
|   NOTABLE COMPONENTS
|
|--tools ..... Other server-components with special build
↪ needs.
|   |--build.xml ..... Build scripts
|   |
|   \--<tool-name>
|       |--build.xml ..... Build file
|       \--ivy.xml ..... Jar dependencies
|
|--antlib ..... Special component which is not built, but
↪ referenced by the build
|   |--resources ..... Build resources
|       |--global.xml ..... Global build properties
|       |--hibernate.xml
|       |--lifecycle.xml ..... Ivy-related targets
|       \--version.xml ..... Version properties
```

Note: User examples are explained under *Working with OMERO*

Unfortunately, just the above snapshot of the code repository omits some of the most important code. Many megabytes of source code is generated both by our own `DSLTask` as well as by the `Ice slice2java`, `slice2cpp`, and `slice2py` code generators. These take an intermediate representation of the `OME-Model` and generate our *OME-Remote Objects*. This code is not available in git, but once built, can be found in all the directories named “generated”.

Build tools

Ant

`./build.py` is a complete replacement for your local ant install. In many cases, you will be fine running **ant**. If you have any issues (for example `OutOfMemory`) , please use `./build.py` instead. **However, only use one or the other; do not mix calls between the two.**

The main build targets are defined in the top-level `build.xml` file. All available targets can be listed using:

```
./build.py -p
```

Ivy

The build system uses `Ivy 2.3.0` as the dependency manager. The general Ivy configuration is defined in a `settings` file located under <https://github.com/ome/openmicroscopy/blob/develop/etc/ivysettings.xml>.

In order to determine the transitive closure of all dependencies, Ivy resolves each `ivy.xml` file and stores the resolved artifacts in a `cache` to speed up other processes. The OMERO build system defines and uses two kinds of caches:

1. the local dependencies cache under `lib/cache` is used by most resolvers
2. Maven resolvers use the Maven cache under `~/.m2/repository`

Note: When the Ivy configuration file or the version number is changed, the cache can become stale. Calling `./build.py clean` from the top-level build will delete the content of the local cache.

`Resolvers` are key to how Ivy functions. Multiple dependency resolvers can be defined fine-grained enough to resolve an individual jar in order to pick up the latest version of any library from a `repository`, a `generic URL` or from the `local file system`. Since OMERO 5.1.3, the remote repository resolvers are set up to resolve transitive dependencies.

The OMERO build system uses by default a `chain resolver` called `omero-resolver` which resolves the following locations in order:

1. `target/repository` which contains most artifacts published by the build system in the *install* step of the lifecycle
2. the local dependency repository under `lib/repository`
3. the local Maven cache under `~/.m2/repository`
4. the `Maven central repository`
5. the `OME artifactory`

Bio-Formats dependencies are resolved using a specific `chain resolver` called `ome-resolver` which resolves the following locations in order:

1. the local Maven cache under `~/.m2/repository`
2. the [OME artifactory](#)

To define its dependencies, each component uses a top-level [Ivy file](#), `ivy.xml`, for the build and optionally another Ivy file, `test.xml`, for the tests.

The OMERO build system defines and uses four types of [Ivy configurations](#):

1. `build`: defines dependencies to be used for building
2. `server`: defines dependencies to be bundled under `lib/server`
3. `client`: defines dependencies to be bundled under `lib/client`
4. `test`: defines dependencies to be used for running the tests

While building, most Java components follow the same lifecycle define in [lifecycle.xml](#). The default `dist` target for each component calls each of the following steps in order:

1. `retrieve`: [retrieve](#) the resolved dependencies and copy them under `target/libs`
2. `prepare`: prepare various resources (property files, <https://github.com/ome/openmicroscopy/blob/develop/lib/logback-build.xml>)
3. `generate`: copy all resources from the previous step for compilation
4. `compile`: compile the source files into the destination repository
5. `package-extra`: package the sources and the Javadoc into Jar files for publication
6. `package`: package the compiled classes into a Jar file for publication
7. `install`: convert the component Ivy file into a pom file using [makepom](#) and [publish](#) the component artifacts

Individual components can override the content of this default lifecycle via their `build.xml`.

OmeroTools

The [Ant](#) build alone is not enough to describe all the products which get built. Namely, the builds for the non-Java components stored under <https://github.com/ome/openmicroscopy/tree/develop/components/tools> are a bit more complex. Each tools component installs its artifacts to the `tools/target` directory which is copied **on top of** the `dist` top-level distribution directory.

Jenkins

The OME project currently uses [Jenkins](#) as a continuous integration server available [here](#), so many binary packages can be downloaded without compiling them yourself. See the [Continuous Integration documentation](#) for further details.

Server build

The default ant target (`build-default`) will build the OMERO system and copy the necessary components for a binary distribution to the `dist` directory. Below is a comparison of what is taken from the build, where it is put, and what role it plays in the distribution.

OMERO_SOURCE_PREFIX	OMERO_SOURCE_PREFIX/dist	Comments
components/tools/OmeroCpp/lib*	lib/	Native shared libraries
lib/repository/<some>	lib/client & lib/server	Libraries needed for the build
etc/	etc/	Configuration
sql/*.sql	sql/	SQL scripts to prepare the database

Note: By default, *OMERO C++ language bindings* are not built. Use `build-all` for that.

These files are then zipped to `OMERO.server-<version>.zip` via `release-zip`

Coupled development

Since OMERO 5.1.3, Bio-Formats is decoupled from the OMERO build system which consumes Bio-Formats artifacts from the OME Maven repository via [Ivy](#).

While this decoupling matches most of the development use cases, it is sometimes necessary to work on coupled Bio-Formats and OMERO branches especially during breaking changes of the OME Data Model or the Bio-Formats API.

The general rule for coupled branches is to build each component in their dependency order and use the local Maven repository under `~/.m2/repository` to share artifacts.

Building Bio-Formats

From the top-level folder of the Bio-Formats repository,

1. if necessary, adjust the version of Bio-Formats which will be built, installed locally and consumed by OMERO e.g. for 5.2.0-SNAPSHOT:

```
$ ./tools/bump_maven_version.py 5.2.0-SNAPSHOT
```

2. run the Maven command allowing to build and install the artifacts under the local Maven cache:

```
$ mvn clean install
```

Building OMERO

From the top-level folder of the OMERO repository,

1. in <https://github.com/ome/omero-model>, adjust the version of `ome:formats-gpl` in <https://github.com/ome/omero-model/blob/v5.6.13/build.gradle> to the version chosen for the Bio-Formats build
2. publish locally the change using `gradle publishToMavenLocal`

3.1.5 Working with OMERO

This page describes various tools and resources useful for working with the OMERO API, as well as some tips on setting up your working environment. It should be useful to client developers working in any of the supported languages. For language specific info, see the following links: [OMERO Java language bindings](#), [OMERO Python language bindings](#), [OMERO C++ language bindings](#), [OMERO MATLAB language bindings](#).

OMERO.clients

The OMERO model is implemented as a relational PostgreSQL database on the OMERO.server and mapped to code-generated model objects used by the clients in the various supported languages (linked above). The OMERO API consists of a number of services for working with these objects and associated binary data. Typically, clients will use various stateless services to query the OMERO model and then use the stateful services for exchange of binary data or image rendering.

A typical client interaction might have an outline such as:

- Log in to OMERO, obtaining connection and ‘service factory’.
- Use the stateless ‘Query Service’ or ‘Container Service’ to traverse Projects, Datasets and Images.
- Use the stateful ‘Rendering Engine’ or ‘Thumbnail Service’ to view images.
- Use the stateful ‘Raw Pixels Service’ or ‘Raw File Store’ to retrieve pixel or file data for analysis.
- Create new Annotations or other objects and save them with the stateless ‘Update Service’.
- Close stateful services to free resources and close the connection.

OMERO.clients use a common ‘gateway’ to communicate with an OMERO.server installation and allow the user to import, display, edit, and manage server data. The OMERO team has developed a suite of clients (see [OMERO clients overview](#)), but the open source nature of the OMERO project also allows developers to create their own, customized clients. If you are interested in doing this, further information is available on [Developing OMERO clients](#).

OMERO server

Although most interactions with OMERO can be achieved remotely, you will generally find it easier to have the server installed on your development machine, particularly if you are going to be doing a lot of OMERO development. This gives you local access to the database, binary repository, logs etc. and means you can work ‘off-line’.

Even if the server you are connecting to is remote, you will still want to have the server package available locally, so as to give you the command line tools, Python libraries, etc. It is important that all OMERO server and client libraries you use are the same OMERO version.

You may wish to work with the most recent OMERO release, or alternatively you can use the latest development code. Instructions on how to download or check out the code can be found on the main [downloads page](#).

Regular builds of the server are performed by [Jenkins](#) including generated Javadocs. See the [contributing developer continuous integration](#) documentation for more information.

Environment variables

In addition to the install instructions, you might find it useful to set the following variables:

- For Python developers, create a virtual environment and install omero-py.
- Add to your PATH the /bin/ directory of the virtual environment e.g. `OMERO_VENV=/opt/omero/server/venv3` where omero-py is installed - allows you to call the 'omero' command from anywhere

```
export PATH=$PATH:$OMERO_VENV/bin/
```

Now checkout the CLI.

```
$ omero -h
```

Network hopping for laptops

By default OMERO will bind to all available interfaces. On a laptop this has the undesirable effect of requiring an OMERO restart when changing network connections, e.g. from a home to a work network connection. To avoid this, it is possible to bind only on the localhost interface which will never change IP address.

```
$ omero config set Ice.Default.Host 127.0.0.1
# Restart to activate the new setting
$ omero admin restart
```

Note: Be warned, if doing this, it will no longer be possible to connect to the OMERO server instance from anywhere except the local machine.

Database access

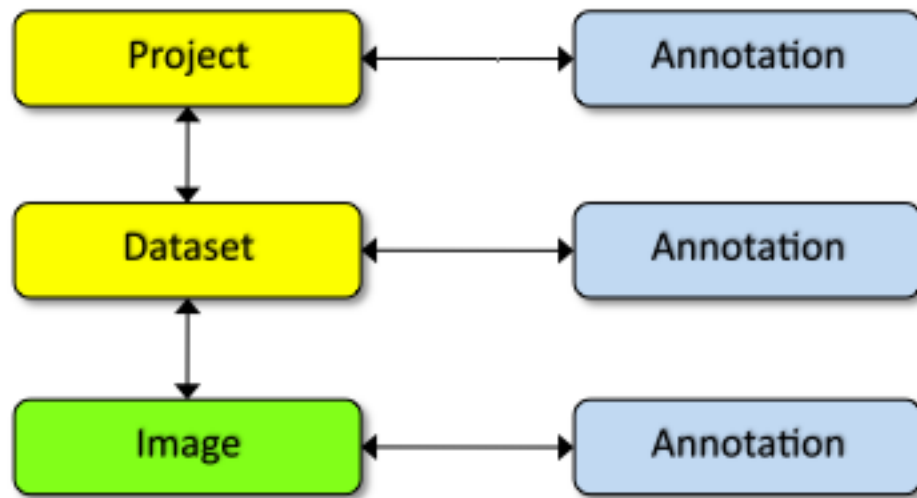
It is useful to be able to directly query or browse the OMERO PostgreSQL database, which can be achieved with a number of tools. E.g.

- `psql` - this command line tool should already be installed. Depending on your permissions, you may need to connect as the 'postgres' user:

```
$ sudo -u postgres psql omero
Password:          # sudo password
omero=# \d;        # give a complete list of tables and views
omero=# \d annotation; # list all the columns in a particular table
omero=# select id, discriminator, ns, textValue, file from annotation order by id_
↵desc;            # query
```

- [pgAdmin](#) is a free, cross platform GUI tool for working with PostgreSQL

OMERO model



You can browse the OMER model in a number of ways, one of which is by looking at the database itself (see above). Another is via the [OMERO model API](#) documentation.

However, due to the complexity of the OMER model, it is helpful to have some starting points (follow links below to the docs themselves).

Note: These figures show highly simplified outlines of various model objects.

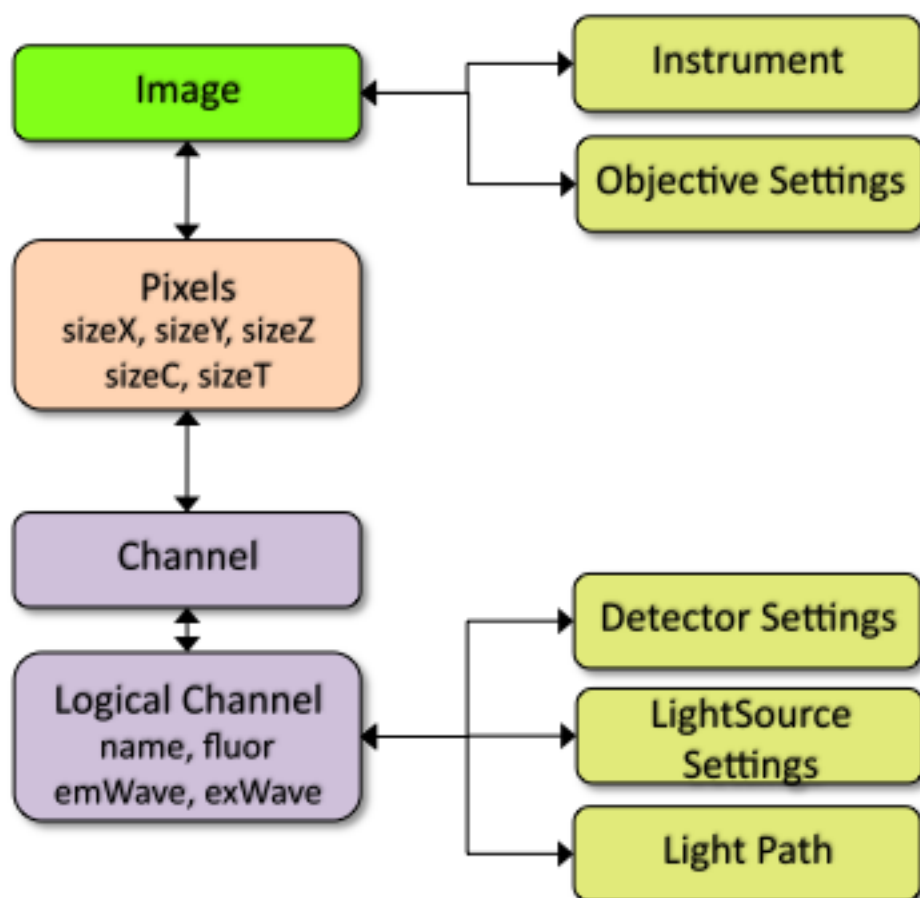
Projects, datasets and images

[Projects](#) and [Datasets](#) are many-to-many containers for [Images](#) (linked by [ProjectDatasetLinks](#) and [DatasetImageLinks](#) respectively).

[Projects](#), [Datasets](#), [Images](#) and a number of other entities can be linked to [Annotations](#) (abstract superclass) via specific links ([ProjectAnnotationLink](#), [DatasetAnnotationLink](#) etc). [Annotation](#) subclasses such as [CommentAnnotation](#), [FileAnnotation](#) etc. are stored in a single database table in OMER (all [Annotations](#) have unique ID).

Images

[Images](#) in OMER are made up of many entities. These include core image components such as [Pixels](#) and [Channels](#), as well as a large number of additional metadata objects such as [Instrument](#) (microscope), [Objective](#), [Filters](#), [Light Sources](#), and [Detectors](#).



Working with the OMERO model objects

For detailed information see *OME-Remote Objects* and *Developing OMERO clients* pages.

Objects that you wish to work with on the client must be loaded from OMERO, with the query defining the extent of any data graph that is “fetched”.

The *OMERO Application Programming Interface* supports two principle ways of querying OMERO and retrieving the objects. You can write SQL-like queries using the query service (uses “HQL”) or you can use one of the other services that already has suitable queries. Using the query service is very flexible but it requires detailed knowledge of the OMERO model (see above) and is susceptible to any change in the model.

For example, to load a specific Project and its linked Datasets you could write a query like this:

```
queryService = session.getQueryService()
params = omero.sys.Parameters()
params.map = {"pid": rlong(projectId)}
query = "select p from Project p left outer join fetch p.datasetLinks as links left
        outer join fetch links.child as dataset where p.id=:pid"
project = queryService.findByQuery(query, params)
for dataset in project.linkedDatasetList:
    print(dataset.getName().getValue())
```

Or use the Container Service like this:

```
containerService = session.getContainerService()
project = containerService.loadContainerHierarchy("Project", [projectId], True)
for dataset in project.linkedDatasetList:
    print(dataset.getName().getValue())
```

For a list of the available services, see the *OMERO Application Programming Interface* page.

Examples

HQL examples

HQL is used for Query Service queries (see above). Some examples, coupled with the references for the *OMERO model* and *HQL syntax* should get you going, along with notes about object loading on the *OME-Remote Objects* page.

Note: If possible, it is advisable to use an existing API method from one of the other services (as for the container service above).

Although it is possible to place query parameters directly into the string, it is preferable (particularly for type-checking) to use the `omero.sys.Parameters` object:

```
queryService.findByQuery("from PixelType as p where p.value='%s'" % pType, None)

# better to do
params = omero.sys.Parameters()
params.map = {"pType": rstring(pType)}
queryService.findByQuery("from PixelType as p where p.value=:pType", params)
```

psql queries

Below are a number of example psql database queries:

```
# list any images that do not have pixels:
omero=#select id, name from Image i where i.id not in (select image from Pixels where
↳image is not null) order by i.id desc;

omero=# select id, name, ome_perms(permissions) from experimentergroup;
id | name | ome_perms
---+---+-----
0 | system | -rw----
1 | user | -rwr-r-
2 | guest | -rw----
3 | JRS-private | -rw----
4 | JRS-read-only | -rwr---
```

```
omero=# select id, name, path, owner_id, group_id, ome_perms(permissions) from
↳originalfile order by id desc limit 100;
id | name | path | owner_id | group_id | ome_perms
---+---+---+---+---+---
56 | GFP-FRAP.cpe.xml | /Users/will/omero/editor/GFP-FRAP.cpe.xml | 4 | 5 | -rwr---
```

```
omero=# \x
Expanded display is on.
omero=# select id, discriminator, ns, textValue, file from annotation where id=369;
-[ RECORD 1 ]-+-----
id | 369
discriminator | /type/OriginalFile/
ns | openmicroscopy.org/omero/import/companionFile
textvalue |
file | 570
```

```
omero=# \x
Expanded display is off.
omero=# select * from joboriginalfilelink where parent = 7;
id | permissions | version | child | creation_id | external_id | group_id | owner_id |
↳update_id | parent
---+---+---+---+---+---+---+---+---
14 | -103 | 7 | 110 | 891 | 208 | 207 |
↳891 | 7
17 | -103 | 7 | 113 | 926 | 208 | 207 |
↳926 | 7
(2 rows)
```

```
omero=# select id, name, path, owner_id, group_id, ome_perms(permissions) from
↳originalfile where id in (110,113) order by id desc limit 100;
id | name | path | owner_id | group_id | ome_perms
---+---+---+---+---+---
```

(continues on next page)

(continued from previous page)

```

-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
113 | stdout | /Users/will/omero/tmp/omero_will/75270/processuLq8fd.dir/out.
↪ | 207 | 208 | -rw----
110 | imagesFromRois.py | ScriptName061ea79c-f98c-447b-b720-d17003d6a72f
↪ | 0 | 0 | -rw----
(2 rows)

# find all annotations on Image ID=2
omero=# select * from annotation where id in (select child from imageannotationlink
↪ where parent = 2) ;

# trouble-shooting postgres
omero=# select * from pg_stat_activity ;

```

omero hql

You can use the **omero hql** command to query a remote OMERO database, entering your login details when requested.

Note: Because you will be querying the database under a particular login, the entries returned will be subject to the permissions of that login.

```

omero hql -q --limit=10 "select name from OriginalFile where id=4106"
omero hql -q --limit=10 "select id, textValue, file from Annotation a order by a.id desc"
omero hql -q --limit=10 "select id, textValue from TagAnnotation a order by a.id desc"
omero hql -q --limit=100 "select id, owner.id, started, userAgent from Session where
↪ closed is null"

```

3.1.6 Running and writing tests

The following guidelines apply to tests in both the Java and Python test components. However, some of the presented options apply to only one or the other.

The default build target does not compile all the required testing resources. You should run *test-compile* (or *build-dev* if you are using Eclipse) first:

```
./build.py build-default test-compile
```

You must rebuild the *test-compile* target if you subsequently modify any of the Java tests.

Note: The OMERO C++ components and tests are under heavy development, and are not compiled or run by the targets mentioned on this page.

Running tests

Running unit tests

Starting from version 5.5, components have been migrated to their own repository.

The following repositories use **Gradle** to run the unit tests:

- <https://github.com/ome/omero-model>
- <https://github.com/ome/omero-common>
- <https://github.com/ome/omero-romio>
- <https://github.com/ome/omero-renderer>
- <https://github.com/ome/omero-server>
- <https://github.com/ome/omero-blitz>
- <https://github.com/ome/omero-gateway-java>

The following repositories use **pytest** to run the unit tests:

- <https://github.com/ome/omero-py>
- <https://github.com/ome/omero-web>

Running integration tests

Integration testing is a bit more complex because of the reliance on a database, which is not easily mockable. All Hibernate-related classes are tested in integration mode.

The tests require a fast computer. Running all the integration tests places several restrictions on the environment:

- There must be a running OMERO database.
- An OMERO.server instance must be running.

Integration tests assume that:

- ICE_CONFIG has been properly set. The contents of the `etc/ice.config` file should be enough to configure a running server for integration testing. This means that code creating a client connection as outlined in *Developing OMERO clients* should execute without errors.
- An OMERO.server instance is running on the host and port specified in the ICE_CONFIG file.

If any of the tests fail with a user authentication exception (or `omero.client` throws an exception), a new `ice.config` file can be created and pointed to by the ICE_CONFIG environment variable. Most likely the first settings that will have to be put there will be `omero.user` and `omero.pass`.

Running all tests

To run all the integration tests, use

```
./build.py test-integration
```

Note that some Python tests are excluded by default, see *Using markers in OmeroPy tests* for more details.

Component tests

Running an integration test suite for an individual component can be done explicitly via:

```
./build.py -f components/<component>/build.xml integration
```

Results are placed in `components/<component>/target/reports`.

Individual tests

Warning: Some integration tests leak file descriptors. If many tests are run then they may start to fail after the system's open files limit is reached. Depending on your system the limit may be checked or adjusted using `ulimit -n` and `/etc/login.conf` or `/etc/security/limits.conf`.

Running Java tests

Individual tests

Alternatively, you can run individual tests which you may currently be working on using the `--tests` parameter. The test class must be provided in the fully qualified name form.

```
cd components/tools/OmeroJava
gradle test --tests "integration.gateway.AdminFacilityTest"
```

Individual test class methods

Individual OmeroJava test class methods can be run using the `--tests` parameter. The test method must be provided in the fully qualified name form.

```
cd components/tools/OmeroJava
gradle test --tests "integration.chgrp.AnnotationMoveTest.testMoveTaggedImage"
```

Individual test groups

To run individual OmeroJava test groups the `--tests` parameter.

```
cd components/tools/OmeroJava
gradle test --tests "integration.*"
```

Using Eclipse to run tests

To facilitate importing OMERO components into Eclipse, there are `.project` and `.classpath-template` files stored in each component directory (e.g. `tools/OmeroJava's .project` and `.classpath-template`).

There are also top-level `.classpath` and `.project` files which allow for importing all components as a single project, but this approach requires more memory and does not clearly differentiate the classpaths, and so can lead to confusion.

Before importing any component as a project into Eclipse, a successful build has to have taken place:

```
./build.py
```

This is for two reasons. Firstly, the Eclipse projects are not configured to perform the code generation needed. The **build.py** command creates the directory:

```
<component>/target
```

which will be missing from any Eclipse project you open before building the source.

Secondly, Ivy is used to copy all the jar dependencies from `OMERO_SOURCE_PREFIX/lib/repository` to `<component>/target/libs`, which is then used in the Eclipse `.classpath` files.

If Eclipse ever gets out of sync after the first build, **./build.py build-eclipse** can be used to quickly synchronize.

A prerequisite of running unit and integration tests in the Eclipse UI is having the TestNG plug-in installed and working (help available on the [TestNG site](#)).

Running the unit tests under Eclipse requires no extra settings and is as easy as navigating to the package or class context menu *Run As* or *Debug As*, then selecting *TestNG*.

Integration tests require the `ICE_CONFIG` environment variable to be available for the Eclipse-controlled JVM. This can be done by editing Debug/Run configurations in Eclipse. After navigating to the Debug (or Run) Configurations window, the *Environment* tab needs to be selected. After clicking *New*, `ICE_CONFIG` can be defined as a path to the `ice.config` file. This setting needs to be defined per package, class or method.

By using the “debug” target from `templates.xml`, it is possible to have OMERO listen on port 8787 for a debugging connection.

```
omero admin stop
omero admin start debug
```

Then in Eclipse, you can create a new “Debug” configuration by clicking on *Remote Java Application*, and setting the port to 8787. These values are arbitrary and can be changed locally.

Keep in mind:

- The server will not start up until you have connected with Eclipse. This is due to the “suspend=y” clause in `templates.xml`. If you would like the server to start without you connecting, use “suspend=n”.
- If you take too much time examining your threads, your calls may throw timeout exceptions.

Running Python tests

Using markers in OmeroPy tests

Tests under OmeroPy can be included or excluded according to markers defined in the tests. This can be done by using the `-DMARK` option. For example, to run all the integration tests marked as `broken`:

```
./build.py -f components/tools/OmeroPy/build.xml integration -DMARK=broken
```

By default tests marked as `broken` are excluded so the following two builds are equivalent:

```
./build.py -f components/tools/OmeroPy/build.xml integration
./build.py -f components/tools/OmeroPy/build.xml integration -DMARK="not broken"
```

In order to run **all** tests, including `broken`, an empty marker must be used:

```
./build.py -f components/tools/OmeroPy/build.xml integration -DMARK=
```

See also:

Marking OmeroPy tests

Running tests directly

When writing tests it can be more convenient, flexible and powerful to run the tests from <https://github.com/ome/openmicroscopy/tree/develop/components/tools/OmeroPy> or <https://github.com/ome/openmicroscopy/tree/develop/components/tools/OmeroWeb> using **pytest**. Since Python is interpreted, tests can be written and then run without having to rebuild or restart the server. A few basic options are shown below.

First create a python virtual environment as described on the *OMERO Python* page, including `omero-py` and `omero-web` if you want to run OmeroWeb tests. Some tests also require the installation of PyTables.

Then install some additional test dependencies:

```
$ pip install pytest pytest-mock pyyaml tables
# for Omeroweb tests
$ pip install pytest-django
```

Run tests directly with `pytest`, setting the `ICE_CONFIG` as described above. Also set `OMERODIR` to point to the Omero.server:

```
export ICE_CONFIG=/path/to/openmicroscopy/etc/ice.config
export OMERODIR=/path/to/OMERO.server-x.x.x-ice36-bxx

cd components/tools/OmeroPy
pytest test/integration/test_admin.py

# OR for OmeroWeb tests:
cd components/tools/OmeroWeb

pytest test/integration/test_annotate.py
```


-k <string>

This option will run all integration tests containing the given string in their names. For example, to run all the tests under `test/integration` with *permissions* in their names:

```
pytest test/integration -k permissions
```

This option can also be used to run a named test within a test module:

```
pytest test/integration/test_admin.py -k testGetGroup
```

-m <marker>

This option will run integration tests depending on the markers they are decorated with. Available markers can be listed using the `pytest --markers` option. For example, to run all integration tests excluding those decorated with the marker *broken*:

```
pytest test/integration -m "not broken"
```

--markers

This option lists available markers for decorating tests:

```
pytest --markers
```

-s

This option allows the standard output to be shown on the console:

```
pytest test/integration/test_admin.py -s
```

-h, --help

This option displays the full list of available options:

```
pytest -h
```

See <https://docs.pytest.org/en/latest/how-to/usage.html> for more help in running tests.

Failing tests

The `test.with.fail` ant property is set to `false` by default, which prevents test failures from failing the build. However, it can instead be set to `true` to allow test failures to fail the build. For example:

```
./build.py -Dtest.with.fail=true integration
```

Some components might provide individual targets for specific tests (e.g. OmeroJava provides the `broken` target for running broken tests). The `build.xml` file is the reference in each component.

Writing tests

Writing Java tests

For more information on writing tests in general see <https://testng.org/>. For a test to be an “integration” test, place it in the “integration” TestNG group. If a test is temporarily broken, add it to the “broken” group:

```
@Test(groups = {"integration", "broken"})
public void testMyStuff() {

}
```

Tests should be of the **Acceptance Test** form. The ticket number for which a test is being written should be added in the TestNG annotation:

```
@Test(groups = "ticket:60")
```

This works at either the method level (see [SetsAndLinksTest.java](#)) or the class level (see [UniqueResultTest.java](#)).

The tests under <https://github.com/ome/openmicroscopy/tree/develop/components/tools/OmeroJava/test> will be the starting point for most Java-client developers coming to OMERO. An example skeleton for an integration test looks similar to

```
@Test(groups = "integration")
public class MyTest {

    omero.client client;

    @BeforeClass
    protected void setup() throws Exception {
        client = new omero.client();
        client.createSession();
    }

    @AfterClass
    protected void tearDown() throws Exception {
        client.closeSession();
    }

    @Test
    public void testSimple() throws Exception {
        client.getSession().getAdminService().getEventContext();
    }

}
```

Writing Python tests

To write and run Python tests you first need to install *pytest*:

```
pip install pytest
```

For more information on writing tests in general see <https://pytest.org/>.

Unit tests can be found in various repositories such as *omero-py*, *omero-web*, and *omero-dropbox*.

Integration tests which require OMERO.server to run are found in the *openmicroscopy* repository. See: <https://github.com/ome/openmicroscopy/tree/develop/components/tools/OmeroPy/test>, <https://github.com/ome/openmicroscopy/tree/develop/components/tools/OmeroWeb/test> and <https://github.com/ome/openmicroscopy/tree/develop/components/tools/OmeroFS/test>.

The file names must begin with *test_* for the tests to be found by *pytest*.

```
import omero.clients

class TestExample(object)

    def setup_method(self, method):
        client = new omero.client()
        client.createSession()

    def teardown_method(self, method):
        client.closeSession()

    def testSimple():
        ec = client.getSession().getAdminService().getEventContext()
        assert ec, "No EventContext!"
```

Marking OmeroPy tests

Methods, classes and functions can be decorated with *pytest* markers to allow for the selection of tests. *pytest* provides some predefined markers and markers can be simply defined as they are used. However, to centralize the use of custom markers they should be defined in <https://github.com/ome/openmicroscopy/tree/develop/components/tools/pytest.ini>.

To view all available markers the *pytest --markers* option can be used with **pytest** or **py.test** as detailed in *Running tests directly*.

There is one custom marker defined:

broken

Used to mark broken tests. These are tests that fail consistently with no obvious quick fix. Broken tests are excluded from the main integration builds and instead are run in a separate daily build. *broken* markers should have a reason, an associated Trac ticket number or both. If there are multiple associated tickets then a comma-separated list should be used.

```
import pytest

class TestExample2(object):

    @pytest.mark.broken(reason="Asserting false", ticket="12345,67890")
```

(continues on next page)

(continued from previous page)

```
def testBroken():
    assert False, "Bound to fail"
```

Using the Python test library

The [OMERO Python test library](#) defines an abstract `ITest` class that implements the connection set up as well as many methods shared amongst all Python integration tests.

Each concrete instance of the `ITest` will initiate a connection to the server specified by the `ICE_CONFIG` environment variable at the `setup_class()` level. The following objects are created by `ITest.setup_class()` and shared by all test methods of this class:

- `self.root` is a client for the root user
- `self.group` is a new group which permissions are set to `ITest.DEFAULT_PERMS` by default. Overriding `DEFAULTS_PERMS` in a subclass of `ITest` means the group will be created with the new permissions.
- `self.user` is a new user and member of `self.group`
- `self.client` is a client for the `self.user` created at class setup.

Additionally, for the `self.client` object, different shortcuts are available:

- `self.sf` is the non-root client session
- `self.update` is the update service for the non-root client session
- `self.query` is the query service for the non-root client session
- `self.ctx` is the event context for the non-root client session. Note this corresponds to the context at creation time and should be refreshed if the context is modified.

The example below inherits the `ITest` class and would create a read-write group by default

```
from omero.testlib import ITest

class TestExample(ITest):

    DEFAULT_PERMS = 'rwrw--' # Override default permissions
    def test1():
        doAction(self.sf)
```

New user and groups can be instantiated by individual tests using the `ITest.new_user()` and `ITest.new_group()` methods:

```
def testNewGroupOwner():
    new_group = self.new_group(perms='rwa---')
    new_owner = self.new_use(group=new_group, owner=True)
    assert new_owner.id.val, "No EventContext!"
```

New clients can be instantiated by individual tests using the `ITest.new_client()` or `ITest.new_client_and_user()` methods:

```
def testNewClient():
    new_client = self.new_user_and_client()
    ec = new_client.getSession().getAdminService().getEventContext()
    assert ec, "No EventContext!"
```

Images can be imported using the `ITest.import_fake_file()` method:

```
def testFileset():
    # 2 images sharing a fileset
    images = self.import_fake_file(2)
    assert len(images) == 2
```

Writing OMERO.web tests

For OMERO.web integration tests, the `OMERO.web` test library defines an abstract `IWebTest` class that inherits from `ITest` and also implements Django clients at the class setup using the `Django testing tools`.

On top of the elements created by `ITest.setup_class()`, the `IWebTest` class creates:

- `self.django_root_client` is a Django test client for the root user
- `self.django_client` is a client for the new user created at the class setup.

```
from omeroweb.testlib import IWebTest

class TestExample(IWebTest):
    def testSimple():
        self.django_client.post('/login/', {'username': 'john'})
```

New Django test clients can be instantiated by individual tests using the `IWebTest.new_django_client()` method:

```
def testNewDjangoClient():
    new_user = self.new_user()
    omeName = new_user.omeName.val
    new_django_client = self.new_django_client(omeName, omeName)
```

See also:

`test_simple.py`

Example test class using the OMERO.web test library methods

3.2 Using the OMERO API

3.2.1 OMERO Python language bindings

To access the OMERO.server Python API, you need to install the Python client libraries.

From OMERO 5.6.0 release, the client library `omero-py` supports Python 3 and is now available on [PyPI](#) and [Conda](#). The `omero-py` API documentation is available at <https://omero-py.readthedocs.io/>. We recommend you use a Python virtual environment to install the client library. You can create one using either `venv` or `conda` (preferred). If you opt for [Conda](#), you will need to install it first, see [miniconda](#) for more details.

To install `omero-py` using `venv`:

```
$ python3 -m venv myenv
$ . myenv/bin/activate
$ pip install omero-py==5.19.0
```

To install `omero-py` using `conda` (preferred):

```
conda create -n myenv -c conda-forge python=3.8 omero-py
conda activate myenv
```

You can then start using the library in the terminal where the environment has been activated:

```
$ python
>>> from omero.gateway import BlitzGateway
>>> conn = BlitzGateway('username', 'password', host='omero.server', port=4064)
>>> conn.connect()
```

In addition to the auto-generated Python libraries of the core *OMERO Application Programming Interface*, `omero-py` includes a more user-friendly Python module ‘BlitzGateway’ that facilitates several aspects of working with the Python API, such as connection handling, object graph traversal and lazy loading.

Building on the ‘BlitzGateway’, the Jackson Laboratory has created a module of convenience functions called *ezomero*.

This page gives you a large number of code samples to get you started. Then we describe a bit more about *Blitz Gateway documentation*.

All the code examples below can be found at <https://github.com/ome/openmicroscopy/tree/develop/examples/Training/python>.

Code samples

Connect to OMERO

- **Import OMERO and the BlitzGateway**

```
import omero.clients
from omero.gateway import BlitzGateway
```

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
```

- **Create a secure-only connection**

By default, once we have logged in, data transfer is not encrypted. To ensure all data is transferred securely pass the secure flag.

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT, secure=True)
conn.connect()
```

- **Create a connection using a context manager**

The BlitzGateway should be closed after use to free up server resources. This can be automatically done by using it as a context manager. This also automatically calls `connect()`.

```
with BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT, secure=True) as conn:
    for p in conn.getObjects('Project'):
        print(p.name)
    ...
# conn.close() is automatically called
```

- **Create a connection using an existing session**

The BlitzGateway can also be initialized from an existing `omero.client` object.

```
>>> client = omero.client(HOST, PORT)
>>> session = client.createSession(USERNAME, PASSWORD)
>>> conn = BlitzGateway(client_obj=client)
```

In this example the BlitzGateway and client will not be closed automatically. If nothing else is using the client object you could use with `BlitzGateway(client_obj=client)` as `conn`.

- **Current session details**

*# By default, you will have logged into your 'current' group in OMERO. This
can be changed by switching group in the OMERO.insight or OMERO.web clients.*

```
user = conn.getUser()
print("Current user:")
print("  ID:", user.getId())
print("  Username:", user.getName())
print("  Full Name:", user.getFullName())

# Check if you are an Administrator
print("  Is Admin:", conn.isAdmin())
if not conn.isFullAdmin():
    # If 'Restricted Administrator' show privileges
    print(conn.getCurrentAdminPrivileges())

print("Member of:")
for g in conn.getGroupsMemberOf():
    print("  ID:", g.getId(), " Name:", g.getName())
group = conn.getGroupFromContext()
print("Current group: ", group.getName())

# List the group owners and other members
owners, members = group.groupSummary()
print("  Group owners:")
for o in owners:
    print("    ID: %s %s Name: %s" % (
        o.getId(), o.getOmeName(), o.getFullName()))
print("  Group members:")
for m in members:
    print("    ID: %s %s Name: %s" % (
        m.getId(), m.getOmeName(), m.getFullName()))

print("Owner of:")
for g in conn.listOwnedGroups():
    print("  ID: ", g.getId(), " Name:", g.getName())

# Added in OMERO 5.0
print("Admins:")
for exp in conn.getAdministrators():
    print("  ID: %s %s Name: %s" % (
        exp.getId(), exp.getOmeName(), exp.getFullName()))
```

(continues on next page)

(continued from previous page)

```
# The 'context' of our current session
ctx = conn.getEventContext()
# print(ctx)      # for more info
```

- **Close connection**

If you did not use the context manager close the session to free up server resources.

```
conn.close()
```

Read data

```
def print_obj(obj, indent=0):
    """
    Helper method to display info about OMERO objects.
    Not all objects will have a "name" or owner field.
    """
    print("""%s%s:%s  Name:"%s" (owner=%s)""" % (
        " " * indent,
        obj.OMERO_CLASS,
        obj.getId(),
        obj.getName(),
        obj.getOwnerOmeName()))
```

- **List all Projects available to me, and their Datasets and Images**

```
# Load first 5 Projects, filtering by default group and owner
my_exp_id = conn.getUser().getId()
default_group_id = conn.getEventContext().groupId
for project in conn.getObjects("Project", opts={'owner': my_exp_id,
                                                'group': default_group_id,
                                                'order_by': 'lower(obj.name)',
                                                'limit': 5, 'offset': 0}):

    print_obj(project)
    # We can get Datasets with listChildren, since we have the Project already.
    # Or conn.getObjects("Dataset", opts={'project', id}) if we have Project ID
    for dataset in project.listChildren():
        print_obj(dataset, 2)
        for image in dataset.listChildren():
            print_obj(image, 4)
```

- **Get Objects by their ID or attributes**

The first argument for `conn.getObjects()` or `conn.getObject()` is the object type. This is not case sensitive. Supported types are project, dataset, image, screen, plate, plateacquisition, acquisition, well, roi, shape, experimenter, experimentergroup, originalfile, fileset, annotation. You can find attributes of these objects at [OMERO model API](#).

```
# Find objects by ID. NB: getObjects() returns a generator, not a list
projects = conn.getObjects("Project", [1, 2, 3])

# Get a single object by ID. Can use "Annotation" for all types of annotations by ID
```

(continues on next page)

(continued from previous page)

```

annotation = conn.getObject("Annotation", 1)

# Find an Object by attribute. E.g. 'name'
images = conn.getObjects("Image", attributes={"name": name})

```

- **Get different types of Annotations***

Supported types are: tagannotation, longannotation, booleanannotation, fileannotation, doubleannotation, termannotation, timestampannotation, mapannotation

```

# List All Tags that you have permission to access
conn.getObjects("TagAnnotation")

# Find Tags with a known text value
tags = conn.getObjects("TagAnnotation", attributes={"textValue": text})

```

- **Retrieve 'orphaned' objects**

```

# We can use the 'orphaned' filter to find Datasets, Images
# or Plates that are not in any parent container
print("\nList orphaned Datasets: \n", "=" * 50)
datasets = conn.getObjects("Dataset", opts={'orphaned': True})
for dataset in datasets:
    print_obj(dataset)

```

- **Retrieve objects in a container**

```

# We can filter Images by their parent Dataset
# We can also filter Datasets by 'project', Plates by 'screen',
# Wells by 'plate'
print("\nImages in Dataset:", datasetId, "\n", "=" * 50)
for image in conn.getObjects('Image', opts={'dataset': datasetId}):
    print_obj(image)

```

- **Retrieve an image by Image ID**

```

# Pixels and Channels will be loaded automatically as needed
image = conn.getObject("Image", imageId)
print(image.getName(), image.getDescription())
# Retrieve information about an image.
print(" X:", image.getSizeX())
print(" Y:", image.getSizeY())
print(" Z:", image.getSizeZ())
print(" C:", image.getSizeC())
print(" T:", image.getSizeT())
# List Channels (loads the Rendering settings to get channel colors)
for channel in image.getChannels():
    print('Channel:', channel.getLabel())
    print('Color:', channel.getColor().getRGB())
    print('Lookup table:', channel.getLut())
    print('Is reverse intensity?', channel.isReverseIntensity())

# render the first timepoint, mid Z section

```

(continues on next page)

(continued from previous page)

```

z = image.getSizeZ() / 2
t = 0
rendered_image = image.renderImage(z, t)
# rendered_image.show()           # popup (use for debug only)
# rendered_image.save("test.jpg")  # save in the current folder

```

- **Get Pixel Sizes for the above Image**

```

size_x = image.getPixelSizeX()          # e.g. 0.132
print(" Pixel Size X:", size_x)
# Units support, new in OMERO 5.1.0
size_x_obj = image.getPixelSizeX(units=True)
print(" Pixel Size X:", size_x_obj.getValue(), "(%s)" % size_x_obj.getSymbol())
# To get the size with different units, e.g. Angstroms
size_x_ang = image.getPixelSizeX(units="ANGSTROM")
print(" Pixel Size X:", size_x_ang.getValue(), "(%s)" % size_x_ang.getSymbol())

```

- **Retrieve Screening data**

```

for screen in conn.getObjects("Screen"):
    print_obj(screen)
    for plate in screen.listChildren():
        print_obj(plate, 2)
        plateId = plate.getId()

```

- **Retrieve Wells and Images within a Plate**

```

plate = conn.getObject("Plate", plateId)
print("\nNumber of fields:", plate.getNumberOfFields())
print("\nGrid size:", plate.getGridSize())
print("\nWells in Plate:", plate.getName())
for well in plate.listChildren():
    index = well.countWellSample()
    print(" Well: ", well.row, well.column, " Fields:", index)
    for index in range(0, index):
        print(" Image: ", \
              well.getImage(index).getName(), \
              well.getImage(index).getId())

```

- **List all annotations on an object. Filter for Tags and get textValue**

```

for ann in project.listAnnotations():
    print(ann.getId(), ann.OMERO_TYPE)
    print(" added by ", ann.link.getDetails().getOwner().getOmeName())
    if ann.OMERO_TYPE == omero.model.TagAnnotationI:
        print("Tag value:", ann.getTextValue())

```

- **Get Links between Objects and Annotations**

```

# Find Images linked to Annotation(s), unlink Images from these annotations
# and link them to another Tag Annotation
annotation_ids = [1, 2, 3]
tag_id = 4

```

(continues on next page)

(continued from previous page)

```

for link in conn.getAnnotationLinks('Image', ann_ids=annotation_ids):
    print("Image ID:", link.getParent().id)
    print("Annotation ID:", link.getChild().id)
    # Update the child of the underlying omero.model.ImageAnnotationLinkI
    link._obj.child = omero.model.TagAnnotationI(tag_id, False)
    link.save()

# Find Annotations linked to Object(s), filter by namespace (optional)
for link in conn.getAnnotationLinks('Image', parent_ids=image_ids, ns=namespace):
    print("Annotation ID:", link.getChild().id)

```

Groups and permissions

- We are logged in to our ‘default’ group

```

group = conn.getGroupFromContext()
print("Current group: ", group.getName())

```

- Each group has defined Permissions set

```

group_perms = group.getDetails().getPermissions()
perm_string = str(group_perms)
permission_names = {
    'rw---': 'PRIVATE',
    'rwr---': 'READ-ONLY',
    'rwra--': 'READ-ANNOTATE',
    'rwrw--': 'READ-WRITE'}
print("Permissions: %s (%s)" % (permission_names[perm_string], perm_string))

```

- By default, any query applies to ALL data that we can access in our Current group.

This will be determined by group permissions e.g. in Read-Only or Read-Annotate groups, this will include other users’ data - see *Groups and permissions system*.

```

projects = conn.listProjects()      # may include other users' data
for p in projects:
    print(p.getName(), "Owner: ", p.getDetails().getOwner().getFullName())

```

```

# Will return None if Image is not in current group
image = conn.getObject("Image", imageId)
print("Image: ", image)

```

- For cross-group querying, use ‘-1’

```

conn.SERVICE_OPTS.setOmeroGroup('-1')
image = conn.getObject("Image", imageId)      # Will query across all my groups
print("Image: ", image)
if image is not None:
    print("Group: ", image.getDetails().getGroup().getName())
    print(image.getDetails().getGroup().getId())    # access groupId without loading
→group

```

- To query only a single group (not necessarily your ‘current’ group)

```
group_id = image.getDetails().getGroup().getId()
# This is how we 'switch group' in webclient
conn.SERVICE_OPTS.setOmeroGroup(group_id)
projects = conn.listProjects()
image = conn.getObject("Image", imageId)
print("Image: ", image)
```

- To set (or change) the owner of an object (Admins only)

```
tag_ann = omero.gateway.TagAnnotationWrapper(conn)
tag_ann.setTextValue("Not owned by me")
# update details of the wrapped omero.model.AnnotationI _obj
tag_ann._obj.details.owner = ExperimenterI(userId, False)
tag_ann.save()

# If we want to perform multiple tasks it may be more convenient to
# connect as another user. We can use 'user_conn' exactly as for 'conn'
user = conn.getObject("Experimenter", userId).getName()
user_conn = conn.suConn(user)
# This annotation will be owned by user
map_ann = omero.gateway.MapAnnotationWrapper(user_conn)
map_ann.setNs(namespace)
map_ann.setValue(key_values)
map_ann.save()
# Link will be owned by the user
project.linkAnnotation(map_ann)
user_conn.close()
```

Raw data access

- Retrieve a given plane

```
# Use the pixelswrapper to retrieve the plane as
# a 2D numpy array see [https://github.com/scipy/scipy]
#
# Numpy array can be used for various analysis routines
#
image = conn.getObject("Image", imageId)
size_z = image.getSizeZ()
size_c = image.getSizeC()
size_t = image.getSizeT()
z, t, c = 0, 0, 0 # first plane of the image
pixels = image.getPrimaryPixels()
plane = pixels.getPlane(z, c, t) # get a numpy array.
print("\nPlane at zct: ", z, c, t)
print(plane)
print("shape: ", plane.shape)
print("min:", plane.min(), " max:", plane.max(), \
      "pixel type:", plane.dtype.name)
```

- Retrieve a given stack

```
# Get a Z-stack of tiles. Using getTiles or getPlanes (see below) returns
# a generator of data (not all the data in hand) The RawPixelsStore is
# only opened once (not closed after each plane) Alternative is to use
# getPlane() or getTile() multiple times - slightly slower.
c, t = 0, 0 # First channel and timepoint
tile = (50, 50, 10, 10) # x, y, width, height of tile

# list of [ (0,0,0,(x,y,w,h)), (1,0,0,(x,y,w,h)), (2,0,0,(x,y,w,h))... ]
zct_list = [(iz, c, t, tile) for iz in range(size_z)]
print("\nZ stack of tiles:")
planes = pixels.getTiles(zct_list)
for i, p in enumerate(planes):
    print("Tile:", zct_list[i], " min:", p.min(), \
          " max:", p.max(), " sum:", p.sum())
```

- Retrieve a given hypercube

```
zct_list = []
for z in range(size_z / 2, size_z): # get the top half of the Z-stack
    for c in range(size_c): # all channels
        for t in range(size_t): # all time-points
            zct_list.append((z, c, t))
print("\nHyper stack of planes:")
planes = pixels.getPlanes(zct_list)
for i, p in enumerate(planes):
    print("plane zct:", zct_list[i], " min:", p.min(), " max:", p.max())
```

- Retrieve a histogram

```
# Get a 256 bin histogram for channel 0 and plane z=0/t=0:
hist = image.getHistogram([0], 256, False, 0, 0)
print(hist)
```

Write data

- Create a new Dataset

```
# Use omero.gateway.DatasetWrapper:
new_dataset = DatasetWrapper(conn, omero.model.DatasetI())
new_dataset.setName('Scipy_Gaussian_Filter')
new_dataset.save()
print("New dataset, Id:", new_dataset.id)
# Can get the underlying omero.model.DatasetI with:
dataset_obj = new_dataset._obj

# OR create the DatasetI directly:
dataset_obj = omero.model.DatasetI()
dataset_obj.setName(rstring("New Dataset"))
dataset_obj = conn.getUpdateService().saveAndReturnObject(dataset_obj, conn.SERVICE_OPTS)
dataset_id = dataset_obj.getId().getValue()
print("New dataset, Id:", dataset_id)
```

- Link to Project

```
link = omero.model.ProjectDatasetLinkI()
# We can use a 'loaded' object, but we might get an Exception
# link.setChild(dataset_obj)
# Better to use an 'unloaded' object (loaded = False)
link.setChild(omero.model.DatasetI(dataset_obj.id.val, False))
link.setParent(omero.model.ProjectI(projectId, False))
conn.getUpdateService().saveObject(link)
```

- Annotate Project with a new Tag

```
tag_ann = omero.gateway.TagAnnotationWrapper(conn)
tag_ann.setValue("New Tag")
tag_ann.setDescription("Add optional description")
tag_ann.save()
project = conn.getObject("Project", projectId)
project.linkAnnotation(tag_ann)
```

- Add a Map Annotation (list of key: value pairs)

```
key_value_data = [{"Drug Name", "Monastrol"}, {"Concentration", "5 mg/ml"}]
map_ann = omero.gateway.MapAnnotationWrapper(conn)
# Use 'client' namespace to allow editing in Insight & web
namespace = omero.constants.metadata.NSCLIENTMAPANNOTATION
map_ann.setNs(namespace)
map_ann.setValue(key_value_data)
map_ann.save()
project = conn.getObject("Project", projectId)
# NB: only link a client map annotation to a single object
project.linkAnnotation(map_ann)
```

- Count the number of annotations on one or many objects

```
print(conn.countAnnotations('Project', [projectId]))
```

- List all annotations on an object. Get text from tags

```
for ann in project.listAnnotations():
    print(ann.getId(), ann.OMERO_TYPE)
    print(" added by ", ann.link.getDetails().getOwner().getOmeName())
    if ann.OMERO_TYPE == omero.model.TagAnnotationI:
        print("Tag value:", ann.getTextValue())
```

- How to create a file annotation and link to a Dataset

```
dataset = conn.getObject("Dataset", dataset_id)
# Specify a local file e.g. could be result of some analysis
file_to_upload = "README.txt" # This file should already exist
with open(file_to_upload, 'w') as f:
    f.write('annotation test')
# create the original file and file annotation (uploads the file etc.)
namespace = "my.custom.demo.namespace"
print("\nCreating an OriginalFile and FileAnnotation")
file_ann = conn.createFileAnnfromLocalFile(
    file_to_upload, mimetype="text/plain", ns=namespace, desc=None)
```

(continues on next page)

(continued from previous page)

```
print("Attaching FileAnnotation to Dataset: ", "File ID:", file_ann.getId(), \
      ", ", file_ann.getFile().getName(), "Size:", file_ann.getFile().getSize())
dataset.linkAnnotation(file_ann)      # link it to dataset.
```

- Download a file annotation linked to a Dataset

```
# make a location to download the file. "download" folder.
path = os.path.join(os.path.dirname(__file__), "download")
if not os.path.exists(path):
    os.makedirs(path)
# Go through all the annotations on the Dataset. Download any file annotations
# we find. Filter by namespace is optional
print("\nAnnotations on Dataset:", dataset.getName())
namespace = "my.custom.demo.namespace"
for ann in dataset.listAnnotations(ns=namespace):
    if isinstance(ann, omero.gateway.FileAnnotationWrapper):
        print("File ID:", ann.getFile().getId(), ann.getFile().getName(), \
              "Size:", ann.getFile().getSize())
        file_path = os.path.join(path, ann.getFile().getName())

        with open(str(file_path), 'wb') as f:
            print("\nDownloading file to", file_path, "...")
            for chunk in ann.getFileInChunks():
                f.write(chunk)
        print("File downloaded!")
```

- Load all the file annotations with a given namespace

```
ns_to_include = [namespace]
ns_to_exclude = []
metadataService = conn.getMetadataService()
annotations = metadataService.loadSpecifiedAnnotations(
    'omero.model.FileAnnotation', ns_to_include, ns_to_exclude, None)
for ann in annotations:
    print(ann.getId().getValue(), ann.getFile().getName().getValue())
```

- Get first annotation with specified namespace

```
ann = dataset.getAnnotation(namespace)
print("Found Annotation with namespace: ", ann.getNs())
```

OMERO tables

- Create a name for the Original File (should be unique)

```
from random import random
table_name = "TablesDemo:%s" % str(random())
col1 = omero.grid.LongColumn('Uid', 'testLong', [])
col2 = omero.grid.StringColumn('MyStringColumnInit', '', 64, [])
columns = [col1, col2]
```

- Create and initialize a new table.

```
resources = conn.c.sf.sharedResources()
repository_id = resources.repositories().descriptions[0].getId().getValue()
table = resources.newTable(repository_id, table_name)
table.initialize(columns)
```

- Add data to the table

```
ids = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
strings = ["one", "two", "three", "four", "five",
           "six", "seven", "eight", "nine", "ten"]
data1 = omero.grid.LongColumn('Uid', 'test Long', ids)
data2 = omero.grid.StringColumn('MyStringColumn', '', 64, strings)
data = [data1, data2]
table.addData(data)
orig_file = table.getOriginalFile()
table.close()           # when we are done, close.
```

- Load the table as an original file

```
orig_file_id = orig_file.id.val
# ...so you can attach this data to an object e.g. Dataset
file_ann = omero.model.FileAnnotationI()
# use unloaded OriginalFileI
file_ann.setFile(omero.model.OriginalFileI(orig_file_id, False))
file_ann = conn.getUpdateService().saveAndReturnObject(file_ann)
link = omero.model.DatasetAnnotationLinkI()
link.setParent(omero.model.DatasetI(datasetId, False))
link.setChild(omero.model.FileAnnotationI(file_ann.getId().getValue(), False))
conn.getUpdateService().saveAndReturnObject(link)
```

- Table API

See also:

:slicedoc_blitz: OMERO Tables <omero/grid/Table.html>

```
open_table = resources.openTable(orig_file)
print("Table Columns:")
for col in open_table.getHeaders():
    print("    ", col.name)
rowCount = open_table.getNumberOfRows()
print("Row count:", rowCount)
```

- Get data from every column of the specified rows

```
row_numbers = [3, 5, 7]
print("\nGet All Data for rows: ", row_numbers)
data = open_table.readCoordinates(range(rowCount))
for col in data.columns:
    print("Data for Column: ", col.name)
    for v in col.values:
        print("    ", v)
```

- Get data from every column of the specified rows with slice


```

row_numbers = [3, 5, 7]
print("\nGet All Data for rows with slice: ", row_numbers)
data = open_table.slice(range(len(open_table.getHeaders())), row_numbers)
for col in data.columns:
    print("Data for Column: ", col.name)
    for v in col.values:
        print("    ", v)

```

- Get data from specified columns of specified rows

```

col_numbers = [1]
start = 3
stop = 7
print("\nGet Data for cols: ", col_numbers, \
      " and between rows: ", start, "-", stop)
data = open_table.read(col_numbers, start, stop)
for col in data.columns:
    print("Data for Column: ", col.name)
    for v in col.values:
        print("    ", v)

```

- Get data from specified columns of specified rows with slice

```

col_numbers = [1]
start = 3
stop = 7
print("\nGet Data for cols: ", col_numbers, \
      " and between rows: ", start, "-", stop, \
      " with slice")
data = open_table.slice(col_numbers, range(start, stop))
for col in data.columns:
    print("Data for Column: ", col.name)
    for v in col.values:
        print("    ", v)

```

- Query the table for rows where the 'Uid' is in a particular range

```

query_rows = open_table.getWhereList(
    "(Uid > 2) & (Uid <= 8)", variables={}, start=0, stop=rowCount, step=0)
data = open_table.readCoordinates(query_rows)
for col in data.columns:
    print("Query Results for Column: ", col.name)
    for v in col.values:
        print("    ", v)
open_table.close()           # we're done

```

- In future, to get the table back from Original File

```

orig_table_file = conn.getObject(
    "OriginalFile", attributes={'name': table_name})    # if name is unique
saved_table = resources.openTable(orig_table_file._obj)
print("Opened table with row-count:", saved_table.getNumberOfRows())
saved_table.close()

```

ROIs

- Initialize service

```
updateService = conn.getUpdateService()
from omero.rtypes import rdouble, rint, rstring
```

- Create ROI

```
# We are using the core Python API and omero.model objects here, since ROIs
# are not yet supported in the Python Blitz Gateway.
#
# First we load our image and pick some parameters for shapes
x = 50
y = 200
width = 100
height = 50
image = conn.getObject("Image", imageId)
z = image.getSizeZ() / 2
t = 0
```

```
# We have a helper function for creating an ROI and linking it to new shapes
def create_roi(img, shapes):
    # create an ROI, link it to Image
    roi = omero.model.RoiI()
    # use the omero.model.ImageI that underlies the 'image' wrapper
    roi.setImage(img._obj)
    for shape in shapes:
        roi.addShape(shape)
    # Save the ROI (saves any linked shapes too)
    return updateService.saveAndReturnObject(roi)
```

```
# Another helper for generating the color integers for shapes
# see https://www.openmicroscopy.org/Schemas/Documentation/Generated/OME-2016-06/ome_xsd.
# →html#Color for background
def rgba_to_int(red, green, blue, alpha=255):
    """ Return the color as an Integer in RGBA encoding """
    return int.from_bytes([red, green, blue, alpha],
                          byteorder='big', signed=True)
```

```
# create a rectangle shape (added to ROI below)
print(("Adding a rectangle at theZ: %s, theT: %s, X: %s, Y: %s, width: %s, " +
      "height: %s") % (z, t, x, y, width, height))
rect = omero.model.RectangleI()
rect.x = rdouble(x)
rect.y = rdouble(y)
rect.width = rdouble(width)
rect.height = rdouble(height)
rect.theZ = rint(z)
rect.theT = rint(t)
rect.textValue = rstring("test-Rectangle")
rect.fillColor = rint(rgba_to_int(255, 255, 255, 255))
rect.strokeColor = rint(rgba_to_int(255, 255, 0, 255))
```

```
# create an Ellipse shape (added to ROI below)
ellipse = omero.model.EllipseI()
ellipse.x = rdouble(y)
ellipse.y = rdouble(x)
ellipse.radiusX = rdouble(width)
ellipse.radiusY = rdouble(height)
ellipse.theZ = rint(z)
ellipse.theT = rint(t)
ellipse.textValue = rstring("test-Ellipse")
```

```
# Create an ROI containing 2 shapes on same plane
# NB: OMERO.insight client doesn't support display
# of multiple shapes on a single plane.
# Therefore the ellipse is removed later (see below)
create_roi(image, [rect, ellipse])
```

```
# create an ROI with single line shape
line = omero.model.LineI()
line.x1 = rdouble(x)
line.x2 = rdouble(x+width)
line.y1 = rdouble(y)
line.y2 = rdouble(y+height)
line.theZ = rint(z)
line.theT = rint(t)
line.textValue = rstring("test-Line")
create_roi(image, [line])
```

```
def create_mask(mask_bytes, bytes_per_pixel=1):
    if bytes_per_pixel == 2:
        divider = 16.0
        format_string = "H" # Unsigned short
        byte_factor = 0.5
    elif bytes_per_pixel == 1:
        divider = 8.0
        format_string = "B" # Unsigned char
        byte_factor = 1
    else:
        message = "Format %s not supported"
        raise ValueError(message)
    steps = math.ceil(len(mask_bytes) / divider)
    mask = []
    for i in range(int(steps)):
        binary = mask_bytes[
            i * int(divider):i * int(divider) + int(divider)]
        format = str(int(byte_factor * len(binary))) + format_string
        binary = struct.unpack(format, binary)
        s = ""
        for bit in binary:
            s += str(bit)
        mask.append(int(s, 2))
    return bytearray(mask)
```

```

mask_x = 50
mask_y = 50
mask_h = 100
mask_w = 100
# Create [0, 1] mask
mask_array = numpy.fromfunction(
    lambda x, y: (x * y) % 2, (mask_w, mask_h))
# Set correct number of bytes per value
mask_array = mask_array.astype(numpy.uint8)
# Convert the mask to bytes
mask_array = mask_array.tostring()
# Pack the bytes to a bit mask
mask_packed = create_mask(mask_array, 1)

# Define mask's fill color
from omero.gateway import ColorHolder
mask_color = ColorHolder()
mask_color.setRed(255)
mask_color.setBlue(0)
mask_color.setGreen(0)
mask_color.setAlpha(100)

```

```

# create an ROI with a single mask
mask = omero.model.MaskI()
mask.setTheC(rint(0))
mask.setTheZ(rint(0))
mask.setTheT(rint(0))
mask.setX(rdoubles(mask_x))
mask.setY(rdoubles(mask_y))
mask.setWidth(rdoubles(mask_w))
mask.setHeight(rdoubles(mask_h))
mask.setFill(rint(mask_color.getInt()))
mask.setTextValue(rstring("test-Mask"))
mask.setBytes(mask_packed)
create_roi(image, [mask])

```

```

# create an ROI with single point shape
point = omero.model.PointI()
point.x = rdoubles(x)
point.y = rdoubles(y)
point.theZ = rint(z)
point.theT = rint(t)
point.textValue = rstring("test-Point")
create_roi(image, [point])

```

```

# create an ROI with a single polygon, setting colors and lineWidth
polygon = omero.model.PolygonI()
polygon.theZ = rint(z)
polygon.theT = rint(t)
polygon.fillColor = rint(rgba_to_int(255, 0, 255, 50))
polygon.strokeColor = rint(rgba_to_int(255, 255, 0))
polygon.strokeWidth = omero.model.LengthI(10, UnitsLength.PIXEL)

```

(continues on next page)

(continued from previous page)

```

points = "10,20 50,150 200,200 250,75"
polygon.points = rstring(points)
polygon.textValue = rstring("test-Polygon")
create_roi(image, [polygon])

```

- Retrieve ROIs linked to an Image

```

roi_service = conn.getRoiService()
result = roi_service.findByImage(imageId, None)
for roi in result.rois:
    print("ROI: ID:", roi.getId().getValue())
    for s in roi.copyShapes():
        shape = {}
        shape['id'] = s.getId().getValue()
        shape['theT'] = s.getTheT().getValue()
        shape['theZ'] = s.getTheZ().getValue()
        if s.getTextValue():
            shape['textValue'] = s.getTextValue().getValue()
        if type(s) == omero.model.RectangleI:
            shape['type'] = 'Rectangle'
            shape['x'] = s.getX().getValue()
            shape['y'] = s.getY().getValue()
            shape['width'] = s.getWidth().getValue()
            shape['height'] = s.getHeight().getValue()
        elif type(s) == omero.model.EllipseI:
            shape['type'] = 'Ellipse'
            shape['x'] = s.getX().getValue()
            shape['y'] = s.getY().getValue()
            shape['radiusX'] = s.getRadiusX().getValue()
            shape['radiusY'] = s.getRadiusY().getValue()
        elif type(s) == omero.model.PointI:
            shape['type'] = 'Point'
            shape['x'] = s.getX().getValue()
            shape['y'] = s.getY().getValue()
        elif type(s) == omero.model.LineI:
            shape['type'] = 'Line'
            shape['x1'] = s.getX1().getValue()
            shape['x2'] = s.getX2().getValue()
            shape['y1'] = s.getY1().getValue()
            shape['y2'] = s.getY2().getValue()
        elif type(s) == omero.model.MaskI:
            shape['type'] = 'Mask'
            shape['x'] = s.getX().getValue()
            shape['y'] = s.getY().getValue()
            shape['width'] = s.getWidth().getValue()
            shape['height'] = s.getHeight().getValue()
        elif type(s) in (
            omero.model.LabelI, omero.model.PolygonI):
            print(type(s), " Not supported by this code")
        # Do some processing here, or just print:
        print("    Shape:",)
        for key, value in shape.items():

```

(continues on next page)

(continued from previous page)

```

        print(" ", key, value,)
    print("")

```

- **Get Pixel Intensities for ROIs**

```

result = roi_service.findByImage(imageId, None)
shape_ids = []
for roi in result.rois:
    for s in roi.copyShapes():
        shape_ids.append(s.id.val)
ch_index = 0
# Z/T will only be used if a shape doesn't have Z/T set
the_z = 0
the_t = 0
stats = roi_service.getShapeStatsRestricted(shape_ids, the_z, the_t, [ch_index])
for s in stats:
    print("Points", s.pointsCount[ch_index])
    print("Min", s.min[ch_index])
    print("Mean", s.mean[ch_index])
    print("Max", s.max[ch_index])
    print("Sum", s.max[ch_index])
    print("StdDev", s.stdDev[ch_index])

```

- **Remove shape from ROI**

```

result = roi_service.findByImage(imageId, None)
for roi in result.rois:
    for s in roi.copyShapes():
        # Find and remove the Shape we added above
        if s.getTextValue() and s.getTextValue().getValue() == "test-Ellipse":
            print("Removing Shape from ROI...")
            roi.removeShape(s)
            roi = updateService.saveAndReturnObject(roi)

```

Delete data

- **Delete Project**

```

# You can delete a number of objects of the same type at the same
# time. In this case 'Project'. Use deleteChildren=True if you are
# deleting a Project and you want to delete Datasets and Images.
obj_ids = [project_id]
delete_children = False
conn.deleteObjects(
    "Project", obj_ids, deleteAnns=True,
    deleteChildren=delete_children, wait=True)

```

- **Retrieve callback and wait until delete completes**

```

# This is not necessary for the Delete to complete. Can be used
# if you want to know when delete is finished or if there were any errors

```

(continues on next page)

(continued from previous page)

```

handle = conn.deleteObjects("Project", [project_id])
cb = omero.callbacks.CmdCallbackI(conn.c, handle)
print("Deleting, please wait.")
while not cb.block(500):
    print(".")
err = isinstance(cb.getResponse(), omero.cmd.ERR)
print("Error?", err)
if err:
    print(cb.getResponse())
cb.close(True)      # close handle too

```

- Delete Annotations on an Object

```

i = conn.getObject("Image", image_id)
to_delete = []
# Optionally to filter by namespace
for ann in i.listAnnotations(ns=namespace):
    to_delete.append(ann.id)
conn.deleteObjects('Annotation', to_delete, wait=True)

```

- Remove Annotations from an Object (unlink but don't delete)

```

i = conn.getObject("Image", image_id)
to_delete = []
for ann in i.listAnnotations():
    to_delete.append(ann.link.id)
conn.deleteObjects("ImageAnnotationLink", to_delete, wait=True)

```

Render Images

- Get thumbnail

```

::
from PIL import Image
from io import BytesIO
# Thumbnail is created using the current rendering
# settings on the image
image = conn.getObject("Image", image_id)
img_data = image.getThumbnail()
rendered_thumb = Image.open(BytesIO(img_data))
# rendered_thumb.show() # shows a pop-up rendered
# rendered_thumb.save("thumbnail.jpg")

```

- Get current settings

```

print("Channel rendering settings:")
for ch in image.getChannels():
    # if no name, get emission wavelength or index
    print("Name: ", ch.getLabel())
    print(" Color:", ch.getColor().getHtml())
    print(" Active:", ch.isActive())
    print(" Levels:", ch.getWindowStart(), "-", ch.getWindowEnd())
print("isGreyscaleRenderingModel:", image.isGreyscaleRenderingModel())
print("Default Z/T positions:")
print("    Z = %s, T = %s" % (image.getDefaultZ(), image.getDefaultT()))

```

- Show the saved rendering settings on this image

```
print("Rendering Defs on Image:")
for rdef in image.getAllRenderingDefs():
    img_data = image.getThumbnail(rdefId=rdef['id'])
    print("    ID: %s (owner: %s %s)" % (
        rdef['id'], rdef['owner']['firstName'], rdef['owner']['lastName']))
```

- Render each channel as a separate grayscale image

```
image.setGreyscaleRenderingModel()
size_c = image.getSizeC()
z = image.getSizeZ() / 2
t = 0
for c in range(1, size_c + 1):      # Channel index starts at 1
    channels = [c]                  # Turn on a single channel at a time
    image.setActiveChannels(channels)
    rendered_image = image.renderImage(z, t)
    # rendered_image.show()          # popup (use for debug only)
    rendered_image.save("channel%s.jpg" % c)    # save in the current folder
```

- Turn 3 channels on, setting their colors

```
image.setColorRenderingModel()
channels = [1, 2, 3]
color_list = ['F00', None, 'FFFF00'] # do not change color of 2nd channel
image.setActiveChannels(channels, colors=color_list)
# max intensity projection 'intmean' for mean-intensity
image.setProjection('intmax')
rendered_image = image.renderImage(z, t) # z and t are ignored for projections
# rendered_image.show()
rendered_image.save("all_channels.jpg")
image.setProjection('normal')          # turn off projection
```

- Turn 2 channels on, setting levels of the first one

```
channels = [1, 2]
range_list = [[100.0, 120.2], [None, None]]
image.setActiveChannels(channels, windows=range_list)
# Set default Z and T. These will be used as defaults for further rendering
image.setDefaultZ(0)
image.setDefaultT(0)
# default compression is 0.9
rendered_image = image.renderImage(z=None, t=None, compression=0.5)
rendered_image.show()
rendered_image.save("two_channels.jpg")
```

- Save the current rendering settings & default Z/T

```
image.saveDefaults()
```

- Reset to settings at import time, and optionally save

```
image.resetDefaults(save=True)
```


Create Image

- Create an image from scratch

```
# This example demonstrates the usage of the convenience method
# createImageFromNumpySeq() Here we create a multi-dimensional image from a
# hard-coded array of data.
from numpy import array, int8
import omero
size_x, size_y, size_z, size_c, size_t = 5, 4, 1, 2, 1
plane1 = array(
    [[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [0, 1, 2, 3, 4], [5, 6, 7, 8, 9]],
    dtype=int8)
plane2 = array(
    [[5, 6, 7, 8, 9], [0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [0, 1, 2, 3, 4]],
    dtype=int8)
planes = [plane1, plane2]
```

```
def plane_gen():
    """generator will yield planes"""
    for p in planes:
        yield p
```

```
desc = "Image created from a hard-coded arrays"
i = conn.createImageFromNumpySeq(
    plane_gen(), "numpy image", size_z, size_c, size_t, description=desc,
    dataset=None)
print('Created new Image:%s Name:"%s"' % (i.getId(), i.getName()))
```

- Set the pixel size using units (added in 5.1.0)

Lengths are specified by value and a unit enumeration Here we set the pixel size X and Y to be 9.8 Angstroms

```
from omero.model.enums import UnitsLength
# Re-load the image to avoid update conflicts
i = conn.getObject("Image", i.getId())
u = omero.model.LengthI(9.8, UnitsLength.ANGSTROM)
p = i.getPrimaryPixels()._obj
p.setPhysicalSizeX(u)
p.setPhysicalSizeY(u)
conn.getUpdateService().saveObject(p)
```

- Create an Image from an existing image

```
# We are going to create a new image by passing the method a 'generator' of 2D
# planes This will come from an existing image, by taking the average of 2
# channels.
zct_list = []
image = conn.getObject('Image', imageId)
size_z, size_c, size_t = image.getSizeZ(), image.getSizeC(), image.getSizeT()
dataset = image.getParent()
pixels = image.getPrimaryPixels()
new_size_c = 1
```

```
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..)
    """
    for z in range(size_z):          # all Z sections
        # Illustrative purposes only, since we only have 1 channel
        for c in range(new_size_c):
            for t in range(size_t):  # all time-points
                channel0 = pixels.getPlane(z, 0, t)
                channel1 = pixels.getPlane(z, 1, t)
                # Here we can manipulate the data in many different ways. As
                # an example we are doing "average"
                # average of 2 channels
                new_plane = (channel0 + channel1) / 2
                print("newPlane for z,t:", z, t, new_plane.dtype, \
                      new_plane.min(), new_plane.max())
                yield new_plane
```

```
desc = ("Image created from Image ID: %s by averaging Channel 1 and Channel 2"
        % imageId)
i = conn.createImageFromNumpySeq(
    plane_gen(), "new image", size_z, new_size_c, size_t, description=desc,
    dataset=dataset)
```

Filesets - added in OMERO 5.0

- Get the 'Fileset' for an Image

```
# A Fileset is a collection of the original files imported to
# create an image or set of images in OMERO.
image = conn.getObject("Image", imageId)
fileset = image.getFileset()          # will be None for pre-FS images
fs_id = fileset.getId()
# List all images that are in this fileset
for fs_image in fileset.copyImages():
    print(fs_image.getId(), fs_image.getName())
# List original imported files
for orig_file in fileset.listFiles():
    name = orig_file.getName()
    path = orig_file.getPath()
    print(path, name)
```

- Get Original Imported Files directly from the image

```
# this will include pre-FS data IF images were archived on import
print(image.countImportedImageFiles())
# specifically count Fileset files
file_count = image.countFilesetFiles()
```

(continues on next page)

(continued from previous page)

```
# list files
if file_count > 0:
    for orig_file in image.getImportedImageFiles():
        name = orig_file.getName()
        path = orig_file.getPath()
        print(path, name)
```

- Can get the Fileset using `conn.getObject()`

```
fileset = conn.getObject("Fileset", fs_id)
```

Python OMERO.scripts

It is relatively straightforward to take the code samples above and re-use them in OMERO.scripts. This allows the code to be run on the OMERO server and called from either the OMERO.insight client or OMERO.web by any users of the server. See [OMERO.scripts user guide](#).

3.2.2 Blitz Gateway documentation

This page provides some background information on the OMERO Python client ‘gateway’ (omero.gateway module).

The Blitz Gateway is a Python client-side library that facilitates working with the OMERO API, handling connection to the server, loading of data objects and providing convenience methods to access the data. It was originally designed as part of the [OMERO.web framework](#), to provide connection and data retrieval services to various web clients. However, we encourage its use for all access to the OMERO Python API.

Connection wrapper

The BlitzGateway class (see [API of development code](#)) is a wrapper for the OMERO client and session objects. It provides various methods for connecting to the OMERO server, querying the status or context of the current connection and retrieving data objects from OMERO.

BlitzGateway can be used as a context manager to ensure the underlying client connection is automatically closed.

For examples see [Code samples](#).

Model object wrappers

OMERO model objects, e.g. `omero.model.Project`, `omero.model.Pixels` etc. (see [full list](#)) are code-generated and mapped to the OMERO database schema. They are language agnostic and their data is in the form of `omero.rtypes` as described in [about model objects](#).

To facilitate work in Python, particularly in web page templates, these Python model objects are wrapped in Blitz Object Wrappers. This hides the use of `rtypes`.

```
import omero
from omero.model import ProjectI
from omero.rtypes import rstring
p = ProjectI()
p.setName(rstring("Omero Model Project")) # attributes are all rtypes
print(p.getName().getValue())             # getValue() to unwrap the rtype
```

(continues on next page)

(continued from previous page)

```

print(p.name.val)                                # short-hand

from omero.gateway import ProjectWrapper
project = ProjectWrapper(obj=p)                  # wrap the model.object
project.setName("Project Wrapper")              # Don't need to use rtypes
print(project.getName())
print(project.name)

print(project._obj)                              # access the wrapped object with ._obj

```

These wrappers also have a reference to the BlitzGateway connection wrapper, so they can make calls to the server and load more data when needed (lazy loading).

```

>>> from omero.gateway import BlitzGateway

>>> conn = BlitzGateway("username", "password", host="localhost", port=4064)
>>> conn.connect()

>>> for p in conn.listProjects():                # Initially we just load Projects
...     print(p.getName())
...     for dataset in p.listChildren():        # lazy-loading of Datasets here
...         print(" ", dataset.getName())
...
TestProject
  Aurora-B
tiff stacks
  newTimeStack
  test
siRNAi
  CENP
  live-cell
  survivin
>>> conn.close()

```

Wrapper coverage

The OMERO data model has a large number of objects, not all of which are used by the *OMERO.web framework*. Therefore, the Blitz gateway (which was originally built for this framework) has not yet been extended to wrap every omero.model object with a specific Blitz Object Wrapper. The current list of object wrappers can be found in the omero.gateway module [API](#). As more functionality is provided by the Blitz Gateway, the coverage of object wrappers will increase accordingly.

Access to the OMERO API services

If you need access to API methods that are not provided by the gateway library, you can get hold of the *OMERO Application Programming Interface*.

Note: These services will always work with `omero.model` objects and not the gateway wrapper objects.

The gateway handles creation and reuse of the API services, so that new ones are not created unnecessarily. Services can be accessed using the methods of the underlying *Service Factory* with the Gateway handling reuse as needed. **Stateless** services (those retrieved with `getXXX` methods e.g. `getQueryService`) are always reused for each call, e.g. `conn.getQueryService()` whereas **stateful** services e.g. `createRenderingEngine` may be created each time.

Not all methods of the service factory are currently supported in the gateway. You can get an idea of the currently supported services by looking at the source code under the `_createProxies` method.

Example: `ContainerService` can load Projects and Datasets in a single call to server (no lazy loading)

```
cs = conn.getContainerService()
projects = cs.loadContainerHierarchy("Project", None, None)
for p in projects:                # omero.model.ProjectI
    print(p.getName().getValue())  # need to 'unwrap' rstring
    for d in p.linkedDatasetList():
        print(d.getName().getValue())
```

Stateful services, reconnection, error handling etc.

The Blitz gateway was designed for use in the *OMERO.web framework* and it is not expected that stateful services will be maintained on the client for significant time. There are various error-handling functionalities in the Blitz gateway that will close existing services and recreate them in order to maintain a working connection. If this happens then any stateful services that you have on the client-side will become stale. Our general advice is to create, use and close the stateful services in the shortest practicable time.

```
try:
    image = conn.getObject("Image", image_id)
    # Initializes the Rendering engine and sets rendering settings
    image.setActiveChannels([1, 2], [[20, 300], [50, 500]], ['00FF00', 'FF0000'])
    pil_image = image.renderImage(0, 0)
    # Now we close the rendering engine
    image._re.close

    # Can continue to use the connection until done,
    # then close ALL services:
finally:
    conn.close()
```

Overwriting and extending omero.gateway classes

When working with `omero.gateway` or wrapper classes such as `omero.gateway.ImageWrapper` you might want to add your own functionality or customize an existing one. N.B. The call to `omero.gateway.refreshWrappers()` is important to update the dictionary of classes used by `conn.getObjects()`. This will ensure that instances of your class are returned by `conn.getObjects()`.

```
class MyBlitzGateway (omero.gateway.BlitzGateway):

    def __init__ (self, *args, **kwargs):
        super(MyBlitzGateway, self).__init__(*args, **kwargs)

        ...do something, e.g. add new field...
        self.new_field = 'foo'

    def connect (self, *args, **kwargs):

        rv = super(MyBlitzGateway, self).connect(*args,**kwargs)
        if rv:
            ...do something, e.g. modify new field...
            self.new_field = 'bla'

        return rv

omero.gateway.BlitzGateway = MyBlitzGateway

class MyBlitzObjectWrapper (object):

    annotation_counter = None

    def countAnnotations (self):
        """
        Count on annotations linked to the object and set the value
        on the custom field 'annotation_counter'.

        @return      Counter
        """

        if self.annotation_counter is not None:
            return self.annotation_counter
        else:
            container = self._conn.getContainerService()
            m = container.getCollectionCount(self._obj.__class__.__name__, type(self._
            ↪obj).ANNOTATIONLINKS, [self._oid], None)
            if m[self._oid] > 0:
                self.annotation_counter = m[self._oid]
                return self.annotation_counter
            else:
                return None

class ImageWrapper (MyBlitzObjectWrapper, omero.gateway.ImageWrapper):
    """
    omero_model_ImageI class wrapper overwrite omero.gateway.ImageWrapper
```

(continues on next page)

(continued from previous page)

```

and extends MyBlitzObjectWrapper.
"""

def __prepare__ (self, **kwargs):
    if kwargs.has_key('annotation_counter'):
        self.annotation_counter = kwargs['annotation_counter']

omero.gateway.ImageWrapper = ImageWrapper

# IMPORTANT to update the map of wrappers for 'Image' etc. returned by getObjects("Image")
omero.gateway.refreshWrappers()

```

3.2.3 Command Line Interface as an OMERO development tool

Working with objects

The **omero obj** command allows to create and update OMERO objects. More information can be displayed using `omero obj -h`.

A complete *Glossary of all OMERO Model Objects* is available for reference.

Object creation

The **omero obj new** subcommand allows to create new objects:

```
$ omero obj new Object field=value
```

where *Object* is the type of object to create, e.g. *Dataset* or *ProjectDatasetLink* and *field/value* is a valid key/value pair for the type of object. For example, the following command creates a new screen with a name and a description:

```
$ omero obj new Screen name=Screen001 description="screen description"
```

Object update

The **omero obj update** subcommand allows to update existing objects:

```
$ omero obj update Object:ID field=value
```

where *Object:ID* is the type and the ID of object to update, e.g. *Image:1* or *PlateDatasetLink:10* and *field/value* is a valid key/value pair to update for the specified object.

For example, the following command updates the existing screen of ID 2 with a name and a description:

```
$ omero obj update Screen:2 name=Screen001 description="screen description"
```

Piping output

The output of each **omero obj** command is formatted as *Object:ID* so that the CLI commands can be redirected and piped together. For example, the following set of commands creates a dataset and a project and links them together:

```
$ dataset=$(omero obj new Dataset name=dataset-1)
$ project=$(omero obj new Project name=project-1)
$ omero obj new ProjectDatasetLink parent=$project child=$dataset
```

Extensions

Plugins can be written and put in the `lib/python/omero/plugins` directory. On execution, all plugins in that directory are registered with the CLI.

For testing purposes the `--path` argument can be used to point to other plugin files or directories, too.

Alternatively, plugins can be added to any directory ending with `omero/plugins`. If this directory is part of the `PYTHONPATH` the CLI will automatically include them. An example of such a plugin is [omero-cli-render](#).

Team-supported CLI plugins will be pip-installable. Search for “[omero cli](#)” on [PyPI](#).

Thread-safety

The `omero.cli.CLI` should be considered not thread-safe. A single connection object is accessible from all plugins via `self.ctx.conn(args)`, and it is assumed that changes to this object will only take place in the current thread. The CLI instance itself, however, can be passed between multiple threads, as long as only one accesses it sequentially, possibly via locking.

See also:

Extending OMERO.server

Other extensions to OMERO

Help for any specific CLI command can be displayed using the `-h` argument. See [Command line help](#) for more information.

General notes

- For installation notes see [Installation](#).
- Any command can be produced by symlinking `bin/omero` to a file of the form “`omero-command-arg1-arg2`”. This is useful under `/etc/rc.d` to have a startup script.
- All commands respond to **omero help**.

See also:

Command Line Interface as an OMERO client

User documentation on the Command Line Interface

OMERO.cli as an OMERO admin tool

System Administrator documentation for the Command Line Interface

3.2.4 OMERO Java language bindings

Using the [Ice Java language mapping](#) from [ZeroC](#), OMERO provides access to your data within an *OMERO.blitz* server from Java code.

All the code examples below can be found at <https://github.com/ome/openmicroscopy/tree/develop/examples/Training/java/src/training>.

Writing client apps

To make use of the OMERO Java API and interact with *OMERO.blitz* from your code, a client application needs the Java bindings available on the classpath.

The required `.jar` files can be obtained in a number of ways:

- from the [OME artifactory](#) where all available artifacts and their POM files can be searched using the Web interface
- using the `OMERO.java` ZIP file downloaded from the [Java](#) section of the OMERO download page. The `libs` directory can then be used on the Java classpath (or attached to a project in Eclipse).
- following the example in [minimal-omero-client](#). Please make sure you are using the proper branch of the repository, as that influences the versions of dependencies defined in the Maven POM file.

Extended classpath

To use the importer, you will need more `jar` files. To see all the current requirements, take a look at the builds on [Jenkins](#), or alternatively examine the dependencies in the `build.gradle` files (e.g. <https://github.com/ome/omero-insight/blob/master/build.gradle>).

Java Gateway

The Java [Gateway](#) is a wrapper around the [Ice Java language mapping](#) and the *OMERO Application Programming Interface* which makes it easier to interact with an OMERO server in Java.

The [Gateway](#) is the central object for maintaining the connection to the server, see [Connect to OMERO](#)

Functionality for interacting with the server is encapsulated into different [facilities](#). For an example using the [Browse-Facility](#) to access Projects, Datasets, etc. see [Read data](#).

As the plain Ice objects can be a bit ‘bulky’ to handle, they are usually wrapped into Java [DataObjects](#).

All the code examples below can be found at <https://github.com/ome/openmicroscopy/tree/develop/examples/Training/java/src/training>.

Connect to OMERO

- **Connect to the server.** Remember to close the session.

```
LoginCredentials cred = new LoginCredentials(userName, password, host, port);

// Alternative using args array:
// args = new String[] { "--omero.host=" + hostName, "--omero.port=" + port,
//                        "--omero.user=" + userName, "--omero.pass=" + password };
// LoginCredentials cred = new LoginCredentials(args);
```

(continues on next page)

(continued from previous page)

```
// If you want to join an existing session you can use the session ID as
// user name and a 'null' password:
// LoginCredentials cred = new LoginCredentials(sessionID, null, host, port);

//Create a simple Logger object which just writes
//to System.out or System.err
Logger simpleLogger = new SimpleLogger();

Gateway gateway = new Gateway(simpleLogger);
ExperimenterData user = gateway.connect(cred);

//for every subsequent call to the server you'll need the
//SecurityContext for a certain group; in this case create
//a SecurityContext for the user's default group.
SecurityContext ctx = new SecurityContext(user.getGroupId());
```

- Close connection. IMPORTANT

```
gateway.disconnect();
```

Read data

The BrowseFacility offers methods for browsing within the data hierarchy. A list of examples follows, indicating how to load Project, Dataset, Screen, etc.

- Retrieve the projects owned by the user currently logged in.

If a Project contains Datasets, the Datasets will automatically be loaded.

```
BrowseFacility browse = gateway.getFacility(BrowseFacility.class);

Collection<ProjectData> projects = browse.getProjects(ctx);

Iterator<ProjectData> i = projects.iterator();
ProjectData project;
Set<DatasetData> datasets;
Iterator<DatasetData> j;
DatasetData dataset;
while (i.hasNext()) {
    project = i.next();
    String name = project.getName();
    long id = project.getId();
    datasets = project.getDatasets();
    j = datasets.iterator();
    while (j.hasNext()) {
        dataset = j.next();
        // Do something here
        // If images loaded.
        // dataset.getImages();
    }
}
```

- Retrieve the Datasets owned by the user currently logged in.

```

BrowseFacility browse = gateway.getFacility(BrowseFacility.class);
Collection<DatasetData> datasets = browse.getDatasets(ctx);

Iterator<DatasetData> i = datasets.iterator();
DatasetData dataset;
Set<ImageData> images;
Iterator<ImageData> j;
ImageData image;
while (i.hasNext()) {
    dataset = i.next();
    images = dataset.getImages();
    j = images.iterator();
    while (j.hasNext()) {
        image = j.next();
        //Do something
    }
}

```

- **Retrieve the Images** contained in a Dataset.

```

BrowseFacility browse = gateway.getFacility(BrowseFacility.class);
Collection<ImageData> images = browse.getImagesForDatasets(ctx, Arrays.
    ↪asList(datasetId));

Iterator<ImageData> j = images.iterator();
ImageData image;
while (j.hasNext()) {
    image = j.next();
    // Do something
}

```

- **Retrieve an Image** if the identifier is known.

```

BrowseFacility browse = gateway.getFacility(BrowseFacility.class);
ImageData image = browse.getImage(ctx, imageId);

```

- **Access information about the image** for example to draw it.

The model is as follows: Image-Pixels i.e. to access valuable data about the image you need to use the pixels object. We now only support one set of pixels per image (it used to be more!).

```

PixelsData pixels = image.getDefaultPixels();
int sizeZ = pixels.getSizeZ(); // The number of z-sections.
int sizeT = pixels.getSizeT(); // The number of timepoints.
int sizeC = pixels.getSizeC(); // The number of channels.
int sizeX = pixels.getSizeX(); // The number of pixels along the X-axis.
int sizeY = pixels.getSizeY(); // The number of pixels along the Y-axis.

```

- **Retrieve Screening data** owned by the user currently logged in.

Note that the wells are not loaded.

```

BrowseFacility browse = gateway.getFacility(BrowseFacility.class);
Collection<ScreenData> screens = browse.getScreens(ctx);

```

(continues on next page)

(continued from previous page)

```

Iterator<ScreenData> i = screens.iterator();
ScreenData screen;
Set<PlateData> plates;
Iterator<PlateData> j;
PlateData plate;
while (i.hasNext()) {
    screen = i.next();
    plates = screen.getPlates();
    j = plates.iterator();
    while (j.hasNext()) {
        plate = j.next();
    }
}

```

- **Retrieve Wells within a Plate.**

Given a plate ID, load the wells.

```

BrowseFacility browse = gateway.getFacility(BrowseFacility.class);
Collection<WellData> wells = browse.getWells(ctx, plateId);

Iterator<WellData> i = wells.iterator();
WellData well;
while (i.hasNext()) {
    well = i.next();
    //Do something
}

```

- **Retrieve Annotations.**

Load the MapAnnotations (Key-Value pairs) for the logged-in user.

```

BrowseFacility browse = gateway.getFacility(BrowseFacility.class);
ImageData image = browse.getImage(ctx, imageId);

// load only this user's annotations
List<Long> userIds = new ArrayList<Long>();
userIds.add(this.user.getId());

// load only MapAnnotations
List<Class<? extends AnnotationData>> types = new ArrayList<Class<? extends
↳ AnnotationData>>();
types.add(MapAnnotationData.class);

MetadataFacility metadata = gateway.getFacility(MetadataFacility.class);
List<AnnotationData> annotations = metadata.getAnnotations(ctx, image,
    types, userIds);
for (AnnotationData annotation : annotations) {
    MapAnnotationData mapAnnotation = (MapAnnotationData) annotation;
    List<NamedValue> list = (List<NamedValue>) mapAnnotation
        .getContent();
    System.out.println("\nMapAnnotation ID: "+mapAnnotation.getId());
}

```

(continues on next page)

(continued from previous page)

```

    for (NamedValue namedValue : list)
        System.out.println(namedValue.name + ": " + namedValue.value);
}

```

Raw data access

- **Retrieve a given plane.**

This is useful when you need for example the pixels intensity.

```

try (RawDataFacility rdf = gateway.getFacility(RawDataFacility.class)) {
    PixelsData pixels = image.getDefaultPixels();
    int sizeZ = pixels.getSizeZ();
    int sizeT = pixels.getSizeT();
    int sizeC = pixels.getSizeC();

    Plane2D p;
    for (int z = 0; z < sizeZ; z++)
        for (int t = 0; t < sizeT; t++)
            for (int c = 0; c < sizeC; c++) {
                p = rdf.getPlane(ctx, pixels, z, t, c);
            }
}

```

- **Retrieve a given tile.**

```

try (RawDataFacility rdf = gateway.getFacility(RawDataFacility.class)) {
    PixelsData pixels = image.getDefaultPixels();
    int sizeZ = pixels.getSizeZ();
    int sizeT = pixels.getSizeT();
    int sizeC = pixels.getSizeC();
    int x = 0;
    int y = 0;
    int width = pixels.getSizeX()/2;
    int height = pixels.getSizeY()/2;
    Plane2D p;
    for (int z = 0; z < sizeZ; z++) {
        for (int t = 0; t < sizeT; t++) {
            for (int c = 0; c < sizeC; c++) {
                p = rdf.getTile(ctx, pixels, z, t, c, x, y, width, height);
            }
        }
    }
}

```

- **Retrieve a given stack.**

This is useful when you need the pixels intensity.

```

PixelsData pixels = image.getDefaultPixels();
int sizeT = pixels.getSizeT();
int sizeC = pixels.getSizeC();

```

(continues on next page)

(continued from previous page)

```

long pixelsId = pixels.getId();
RawPixelsStorePrx store = null;
try{
    store = gateway.getPixelsStore(ctx);
    store.setPixelsId(pixelsId, false);
    for (int t = 0; t < sizeT; t++) {
        for (int c = 0; c < sizeC; c++) {
            byte[] plane = store.getStack(c, t);
            //Do something
        }
    }
} finally {
    store.close();
}

```

- **Retrieve a given hypercube.**

This is useful when you need the pixels intensity.

```

PixelsData pixels = image.getDefaultPixels();
long pixelsId = pixels.getId();
//offset values in each dimension XYZCT
List<Integer> offset = new ArrayList<Integer>();
int n = 5;
for (int i = 0; i < n; i++) {
    offset.add(i, 0);
}

List<Integer> size = new ArrayList<Integer>();
size.add(pixels.getSizeX());
size.add(pixels.getSizeY());
size.add(pixels.getSizeZ());
size.add(pixels.getSizeC());
size.add(pixels.getSizeT());

//indicate the step in each direction, step = 1,
//will return values at index 0, 1, 2.
//step = 2, values at index 0, 2, 4 etc.
List<Integer> step = new ArrayList<Integer>();
for (int i = 0; i < n; i++) {
    step.add(i, 1);
}
RawPixelsStorePrx store = null;
try {
    store = gateway.getPixelsStore(ctx);
    store.setPixelsId(pixelsId, false);
    byte[] values = store.getHypercube(offset, size, step);
    //Do something
} finally {
    store.close();
}

```

- **Retrieve a histogram.**

```
try (RawDataFacility rdf = gateway.getFacility(RawDataFacility.class)) {
    PixelsData pixels = image.getDefaultPixels();
    int[] channels = new int[] { 0 };
    int binCount = 256;
    Map<Integer, int[]> histdata = rdf.getHistogram(ctx, pixels,
        channels, binCount, false, null);
    int[] histogram = histdata.get(0);
    //Do something with the histogram data
}
```

Write data

- Create a dataset and link it to an existing project.

```
DataManagerFacility dm = gateway.getFacility(DataManagerFacility.class);

//Using IObject directly
Dataset dataset = new DatasetI();
dataset.setName(omero.rtypes.rstring("new Name 1"));
dataset.setDescription(omero.rtypes.rstring("new description 1"));
ProjectDatasetLink link = new ProjectDatasetLinkI();
link.setChild(dataset);
link.setParent(new ProjectI(projectId, false));
IObject r = dm.saveAndReturnObject(ctx, link);

//Using the pojo
DatasetData datasetData = new DatasetData();
datasetData.setName("new Name 2");
datasetData.setDescription("new description 2");
BrowseFacility b = gateway.getFacility(BrowseFacility.class);
ProjectData projectData = b.getProjects(ctx, Collections.singleton(projectId)).
    →iterator().next();
datasetData.setProjects(Collections.singleton(projectData));
DataObject r2 = dm.saveAndReturnObject(ctx, datasetData);
```

- Import images into a dataset.

Using the Java API directly:

```
String[] paths = new String[] { "/pathTo/image1.dv", "/pathTo/image2.dv" };

ImportConfig config = new ome.formats.importer.ImportConfig();

config.email.set("");
config.sendFiles.set(true);
config.sendReport.set(false);
config.contOnError.set(false);
config.debug.set(false);

config.hostname.set("localhost");
config.port.set(4064);
config.username.set("root");
```

(continues on next page)

(continued from previous page)

```

config.password.set("omero");

// the imported image will go into 'orphaned images' unless
// you specify a particular existing dataset like this:
// config.target.set("Dataset:123");

OMEROMetadataStoreClient store;
try {
    store = config.createStore();
    store.logVersionInfo(config.getIniVersionNumber());
    OMEROWrapper reader = new OMEROWrapper(config);
    ImportLibrary library = new ImportLibrary(store, reader);

    ErrorHandler handler = new ErrorHandler(config);
    library.addObserver(new LoggingImportMonitor());

    ImportCandidates candidates = new ImportCandidates(reader, paths, handler);
    reader.setMetadataOptions(new DynamicMetadataOptions(MetadataLevel.ALL));
    library.importCandidates(config, candidates);

    store.logout();
} catch (Exception e) {
    e.printStackTrace();
}

```

- Create a tag (tag annotation) and link it to an existing project.

```

DataManagerFacility dm = gateway.getFacility(DataManagerFacility.class);

TagAnnotation tag = new TagAnnotationI();
tag.setTextValue(omero.rtypes.rstring("new tag 1"));
tag.setDescription(omero.rtypes.rstring("new tag 1"));

//Using the model object (recommended)
TagAnnotationData tagData = new TagAnnotationData("new tag 2");
tagData.setTagDescription("new tag 2");

ProjectAnnotationLink link = new ProjectAnnotationLinkI();
link.setChild(tag);
link.setParent(new ProjectI(info.getProjectId(), false));
IObject r = dm.saveAndReturnObject(ctx, link);
//With model object
link = new ProjectAnnotationLinkI();
link.setChild(tagData.asAnnotation());
link.setParent(new ProjectI(info.getProjectId(), false));
r = dm.saveAndReturnObject(ctx, link);

```

- Create a map annotation (list of key: value pairs) and link it to an existing project.

```

List<NamedValue> result = new ArrayList<NamedValue>();
result.add(new NamedValue("mitomycin-A", "20mM"));

```

(continues on next page)

(continued from previous page)

```

result.add(new NamedValue("PBS", "10mM"));
result.add(new NamedValue("incubation", "5min"));
result.add(new NamedValue("temperature", "37"));
result.add(new NamedValue("Organism", "Homo sapiens"));
MapAnnotationData data = new MapAnnotationData();
data.setContent(result);
data.setDescription("Training Example");
//Use the following namespace if you want the annotation to be editable
//in the webclient and insight
data.setNameSpace(MapAnnotationData.NS_CLIENT_CREATED);
DataManagerFacility fac = gateway.getFacility(DataManagerFacility.class);
fac.attachAnnotation(ctx, data, new ProjectData(new ProjectI(projectId, false)));

```

- **Create a file annotation and link to an image.**

To attach a file to an object e.g. an image, few objects need to be created:

1. an OriginalFile
2. a FileAnnotation
3. a link between the Image and the FileAnnotation.

```

int INC = 262144;
DataManagerFacility dm = gateway.getFacility(DataManagerFacility.class);

//To retrieve the image see above.
File file = File.createTempFile("temp-file-name_", ".tmp");
String name = file.getName();
String absolutePath = file.getAbsolutePath();
String path = absolutePath.substring(0,
    absolutePath.length()-name.length());

//create the original file object.
OriginalFile originalFile = new OriginalFileI();
originalFile.setName(omero.rtypes.rstring(name));
originalFile.setPath(omero.rtypes.rstring(path));
originalFile.setSize(omero.rtypes.rlong(file.length()));
final ChecksumAlgorithm checksumAlgorithm = new ChecksumAlgorithmI();
checksumAlgorithm.setValue(omero.rtypes.rstring(ChecksumAlgorithmSHA1160.value));
originalFile.setHasher(checksumAlgorithm);
originalFile.setMimetype(omero.rtypes.rstring(fileMimeType)); // or "application/octet-
↳stream"
//Now we save the originalFile object
originalFile = (OriginalFile) dm.saveAndReturnObject(ctx, originalFile);

//Initialize the service to load the raw data
RawFileStorePrx rawFileStore = gateway.getRawFileService(ctx);

long pos = 0;
int rlen;
byte[] buf = new byte[INC];
ByteBuffer bbuf;
//Open file and read stream

```

(continues on next page)

(continued from previous page)

```

try (FileInputStream stream = new FileInputStream(file)) {
    rawFileStore.setFileId(originalFile.getId().getValue());
    while ((rlen = stream.read(buf)) > 0) {
        rawFileStore.write(buf, pos, rlen);
        pos += rlen;
        bbuf = ByteBuffer.wrap(buf);
        bbuf.limit(rlen);
    }
    originalFile = rawFileStore.save();
} finally {
    rawFileStore.close();
}
//now we have an original File in DB and raw data uploaded.
//We now need to link the Original file to the image using
//the File annotation object. That's the way to do it.
FileAnnotation fa = new FileAnnotationI();
fa.setFile(originalFile);
fa.setDescription(omero.rtypes.rstring(description)); // The description set above e.g.
↳ PointsModel
fa.setNs(omero.rtypes.rstring(NAME_SPACE_TO_SET)); // The name space you have set to
↳ identify the file annotation.

//save the file annotation.
fa = (FileAnnotation) dm.saveAndReturnObject(ctx, fa);

//now link the image and the annotation
ImageAnnotationLink link = new ImageAnnotationLinkI();
link.setChild(fa);
link.setParent(image.asImage());
//save the link back to the server.
link = (ImageAnnotationLink) dm.saveAndReturnObject(ctx, link);
// o attach to a Dataset use DatasetAnnotationLink;

```

- Load all the file annotations with a given namespace.

```

long userId = gateway.getLoggedInUser().getId();
List<String> nsToInclude = new ArrayList<String>();
nsToInclude.add(NAME_SPACE_TO_SET);
List<String> nsToExclude = new ArrayList<String>();
ParametersI param = new ParametersI();
param.exp(omero.rtypes.rlong(userId)); //load the annotation for a given user.
IMetadataPrx proxy = gateway.getMetadataService(ctx);
List<Annotation> annotations = proxy.loadSpecifiedAnnotations(
    FileAnnotation.class.getName(), nsToInclude, nsToExclude, param);
//Do something with annotations.

```

- Read the attachment.

First load the annotations, cf. above.

```

Iterator<Annotation> j = annotations.iterator();
Annotation annotation;
FileAnnotationData fa;

```

(continues on next page)

(continued from previous page)

```

RawFileStorePrx store = gateway.getRawFileService(ctx);
File file = File.createTempFile("temp-file-name_", ".tmp");
int index = 0;

OriginalFile of;
IQueryPrx svc = gateway.getQueryService(ctx);

try (FileOutputStream stream = new FileOutputStream(file)) {
    while (j.hasNext()) {
        annotation = j.next();
        if (annotation instanceof FileAnnotation && index == 0) {
            fa = new FileAnnotationData((FileAnnotation) annotation);
            //Load the original file
            of = (OriginalFile) svc.get("OriginalFile", fa.getFileID());
            store.setFileId(fa.getFileID());
            int offset = 0;
            long size = of.getSize().getValue();
            //name of the file
            String fileName = of.getName().getValue();
            try {
                for (offset = 0; (offset+INC) < size;) {
                    stream.write(store.read(offset, INC));
                    offset += INC;
                }
            } finally {
                stream.write(store.read(offset, (int) (size-offset)));
            }
            index++;
        }
    }
} finally {
    store.close();
}
file.delete();

```

How to use OMERO tables

- Create and read a table.

In the following example, we create a table with 2 columns.

```

TableDataColumn[] columns = new TableDataColumn[3];
columns[0] = new TableDataColumn("ID", 0, Long.class);
columns[1] = new TableDataColumn("Name", 1, String.class);
columns[2] = new TableDataColumn("Value", 2, Double.class);

Object[][] data = new Object[3][5];
data[0] = new Long[] {1l, 2l, 3l, 4l, 5l};
data[1] = new String[] {"one", "two", "three", "four", "five"};
data[2] = new Double[] {1d, 2d, 3d, 4d, 5d};

```

(continues on next page)

(continued from previous page)

```

TableData tableData = new TableData(columns, data);

TablesFacility fac = gateway.getFacility(TablesFacility.class);

// Attach the table to the image
tableData = fac.addTable(ctx, image, "My Data", tableData);

// Find the table again
Collection<FileAnnotationData> tables = fac.getAvailableTables(ctx, image);
long fileId = tables.iterator().next().getFileID();

// Request second and third column of the first three rows
TableData tableData2 = fac.getTable(ctx, fileId, 0, 2, 1, 2);

// do something, e.g. print to System.out
int nRows = tableData2.getData()[0].length;
for (int row = 0; row < nRows; row++) {
    for (int col = 0; col < tableData2.getColumns().length; col++) {
        Object o = tableData2.getData()[col][row];
        System.out.print(o + " ["
            + tableData2.getColumns()[col].getType() + "]\t");
    }
    System.out.println();
}

```

ROIs

To learn about the model see the [ROI Model documentation](#). Note that annotations can be linked to ROI or shape.

- **Create ROI.**

In this example, we create an ROI with a rectangular shape and attach it to an image.

```

DataManagerFacility dm = gateway.getFacility(DataManagerFacility.class);
ROIFacility roifac = gateway.getFacility(ROIFacility.class);

//To retrieve the image see above.
ROIData data = new ROIData();
data.setImage(image);
//Create a rectangle.
RectangleData rectangle = new RectangleData(10, 10, 10, 10);
rectangle.setZ(0);
rectangle.setT(0);
data.addShapeData(rectangle);

//Add a mask
PixelsData pixels = image.getDefaultPixels();
long pixelsId = pixels.getId();
RawPixelsStorePrx store = gateway.getPixelsStore(ctx);
try {
    store.setPixelsId(pixelsId, false);
    byte[] mask = store.getStack(0, 0);
}

```

(continues on next page)

(continued from previous page)

```

MaskData maskData = new MaskData(10, 10, 100.0, 100.0, mask);
maskData.setZ(0);
maskData.setT(0);
data.addShapeData(maskData);
} finally {
    store.close();
}

//Create an ellipse.
EllipseData ellipse = new EllipseData(10, 10, 10, 10);
//Not setting the Z and T for this shape object, this is also allowed in the model.
//set angle of rotation
double theta = 10;
//create transform object
AffineTransformI newTform = omero.model.AffineTransformI();
newTform.setA00(omero.rtypes.rdouble(cos(theta)));
newTform.setA10(omero.rtypes.rdouble(-sin(theta)));
newTform.setA01(omero.rtypes.rdouble(sin(theta)));
newTform.setA11(omero.rtypes.rdouble(cos(theta)));
newTform.setA02(omero.rtypes.rdouble(0));
newTform.setA12(omero.rtypes.rdouble(0));
//add transform
ellipse.setTransform(newTform);
data.addShapeData(ellipse);

// Save ROI and shape
ROIData roiData = roifac.saveROIs(ctx, image.getId(), Arrays.asList(data)).iterator().
    ↪next();

//now check that the shape has been added.
//Retrieve the shape on plane (z, t) = (0, 0)
List<ShapeData> shapes = roiData.getShapes(0, 0);
Iterator<ShapeData> i = shapes.iterator();
while (i.hasNext()) {
    ShapeData shape = i.next();
    // plane info
    int z = shape.getZ();
    int t = shape.getT();
    long id = shape.getId();
    if (shape instanceof RectangleData) {
        RectangleData rectData = (RectangleData) shape;
        //Insert code to handle rectangle
    } else if (shape instanceof EllipseData) {
        EllipseData ellipseData = (EllipseData) shape;
        //Insert code to handle ellipse
    } else if (shape instanceof LineData) {
        LineData lineData = (LineData) shape;
        //Insert code to handle line
    } else if (shape instanceof PointData) {
        PointData pointData = (PointData) shape;
        //Insert code to handle point
    } else if (shape instanceof MaskData) {

```

(continues on next page)

(continued from previous page)

```

MaskData maskData1 = (MaskData) shape;
//Insert code to handle mask
}

//Check if the shape has transform
//https://blog.openmicroscopy.org/data-model/future-plans/2016/06/20/shape-transforms/
AffineTransformI transform = shape.getTransform();
if (transform != null){

    double xScaling = transform.getA00.getValue();
    double xShearing = transform.getA01.getValue();
    double xTranslation = transform.getA02.getValue();

    double yScaling = transform.getA11.getValue();
    double yShearing = transform.getA10.getValue();
    double yTranslation = transform.getA12.getValue();
    //Insert code to handle transforms
}
}

```

- Retrieve ROIs linked to an Image.

```

ROIFacility roifac = gateway.getFacility(ROIFacility.class);

//Retrieve the roi linked to an image
List<ROIResult> roireresults = roifac.loadROIs(ctx, image.getId());
ROIResult r = roireresults.iterator().next();
if (r == null) return;
Collection<ROIData> rois = r.getROIs();
List<Shape> list;
Iterator<Roi> j = rois.iterator();
while (j.hasNext()) {
    roi = j.next();
    list = roi.copyShapes();
    // Do something
}

```

- Remove a shape from ROI.

```

DataManagerFacility dm = gateway.getFacility(DataManagerFacility.class);
ROIFacility roifac = gateway.getFacility(ROIFacility.class);

//Retrieve the roi linked to an image
List<ROIResult> roireresults = roifac.loadROIs(ctx, image.getId());
ROIResult r = roireresults.iterator().next();
List<Roi> rois = r.rois;
List<Shape> list;
Iterator<Roi> j = rois.iterator();
while (j.hasNext()) {
    roi = j.next();
    list = roi.copyShapes();
    // remove the first shape.
}

```

(continues on next page)

(continued from previous page)

```

if (list.size() > 0) {
    roi.removeShape(list.get(0));
    // update the roi.
    dm.saveAndReturnObject(ctx, roi).saveAndReturnObject(roi);
}
}

```

- **Organize ROIs in Folders.**

```

ROIFacility roifac = gateway.getFacility(ROIFacility.class);

Collection<ROIData> rois = ...

// Add each ROI to a different folder
for (ROIData r : rois) {
    FolderData folder = new FolderData();
    folder.setName("Folder for ROI " + r.getId());
    roifac.addRoIsToFolders(ctx, image.getId(), Arrays.asList(r),
        Arrays.asList(folder));
}

// Get the ROI folders associated with an image
Collection<FolderData> folders = roifac.getROIFolders(ctx, image.getId());
for (FolderData folder : folders) {
    Collection<ROIResult> result = roifac.loadROIsForFolder(ctx,
        image.getId(), folder.getId());
    Collection<ROIData> folderRoIs = result.iterator().next().getROIs();
    // Do something with the ROIs
}

```

Delete data

It is possible to delete Projects, datasets, images, ROIs etc. and objects linked to them depending on the specified options (see [Deleting in OMERO](#)).

- **Delete Image.**

In the following example, we create an image and delete it.

```

DataManagerFacility dm = gateway.getFacility(DataManagerFacility.class);

//First create an image.
ImageData image = new ImageData();
image.setName("image1");
image.setDescription("descriptionImage1");
IOObject object = dm.saveAndReturnObject(ctx, image.asIOObject());

Response rsp = dm.delete(ctx, object).loop(10, 500);

```

Render Images

- Initialize the rendering engine and render an image.

```
PixelsData pixels = image.getDefaultPixels();
long pixelsId = pixels.getId();
RenderingEnginePrx proxy = null;
proxy = gateway.getRenderingService(ctx, pixelsId);
ByteArrayInputStream stream = null;
try {
    proxy.lookupPixels(pixelsId);
    if (!(proxy.lookupRenderingDef(pixelsId))) {
        proxy.resetDefaultSettings(true);
        proxy.lookupRenderingDef(pixelsId);
    }
    proxy.load();
    //Now can interact with the rendering engine.
    proxy.setActive(0, Boolean.valueOf(false));
    PlaneDef pDef = new PlaneDef();
    pDef.z = 0;
    pDef.t = 0;
    pDef.slice = omero.romio.XY.value;
    //render the data uncompressed.
    int[] uncompressed = proxy.renderAsPackedInt(pDef);
    byte[] compressed = proxy.renderCompressed(pDef);
    //Create a buffered image
    stream = new ByteArrayInputStream(compressed);
    BufferedImage image = ImageIO.read(stream);
} finally {
    proxy.close();
    if (stream != null) stream.close();
}
```

- Retrieve thumbnails.

```
ThumbnailStorePrx store = gateway.getThumbnailService(ctx);
ByteArrayInputStream stream = null;
try {
    PixelsData pixels = image.getDefaultPixels();
    store.setPixelsId(pixels.getId())
    //retrieve a 96x96 thumbnail.
    byte[] array = store.getThumbnail(
        omero.rtypes.rint(96), omero.rtypes.rint(96));
    stream = new ByteArrayInputStream(array);
    //Create a buffered image to display
    ImageIO.read(stream);
} finally {
    store.close();
    if (stream != null) stream.close();
}
```


Create Image

The following example shows how to create an Image from an Image already in OMERO. Similar approach can be applied when uploading an image.

```
//See above how to load an image.
PixelsData pixels = image.getDefaultPixels();
int sizeZ = pixels.getSizeZ();
int sizeT = pixels.getSizeT();
int sizeC = pixels.getSizeC();
int sizeX = pixels.getSizeX();
int sizeY = pixels.getSizeY();
long pixelsId = pixels.getId();

//Read the pixels from the source image.
RawPixelsStorePrx store = gateway.getPixelsStore(ctx);
try{
    store.setPixelsId(pixelsId, false);

    List<byte[]> planes = new ArrayList<byte[]>();

    for (int z = 0; z < sizeZ; z++) {
        for (int t = 0; t < sizeT; t++) {
            planes.add(store.getPlane(z, 0, t));
        }
    }
} finally {
    //Better to close to free space.
    store.close();
}

//Now we are going to create the new image.
IPixelsPrx proxy = gateway.getPixelsService(ctx);

//Search for PixelsType object matching the source image.
List<IObject> l = proxy.getAllEnumerations(PixelsType.class.getName());
Iterator<IObject> i = l.iterator();
PixelsType type = null;
String original = pixels.getPixelType();
while (i.hasNext()) {
    PixelsType o = (PixelsType) i.next();
    String value = o.getValue().getValue();
    if (value.equals(original)) {
        type = o;
        break;
    }
}
if (type == null)
    throw new Exception("Pixels Type not valid.");

//Create new image.
String name = "newImageFrom"+image.getId();
RLong idNew = proxy.createImage(sizeX, sizeY, sizeZ, sizeT, Arrays.asList(0), type, name,
```

(continues on next page)

(continued from previous page)

```

        "From Image ID: "+image.getId());
    if (idNew == null)
        throw new Exception("New image could not be created.");
    IContainerPrx proxyCS = entryUnencrypted.getContainerService();
    List<Image> results = proxyCS.getImages(Image.class.getName(),
        Arrays.asList(idNew.getValue()), new ParametersI());
    ImageData newImage = new ImageData(results.get(0));

    //Link the new image and the dataset hosting the source image.
    DatasetImageLink link = new DatasetImageLinkI();
    link.setParent(new DatasetI(datasetId, false));
    link.setChild(new ImageI(newImage.getId(), false));
    gateway.getUpdateService(ctx).saveAndReturnObject(link);

    //Write the data.
    try {
        store = gateway.getPixelsStore(ctx);
        store.setPixelsId(newImage.getDefaultPixels().getId(), false);
        int index = 0;
        for (int z = 0; z < sizeZ; z++) {
            for (int t = 0; t < sizeT; t++) {
                store.setPlane(planes.get(index++), z, 0, t);
            }
        }

        //Save the data.
        store.save();
    } finally {
        store.close();
    }
}

```

Sudo (working within another user's context)

The next code snippet shows how you can work within another user's context. This could for example be a data analyst doing some analysis on behalf of a user and attaching the results to the user's data. The important point is that the user will be the owner of these results and can work with them as usual. The user and 'analyst' do not have to be member of a read-annotate group (see *OMERO permissions querying, usage and history*), but the 'analyst' has to be a 'light administrator' with 'sudo' permission, see *The server's view of administrator restrictions*.

```

AdminFacility admin = gateway.getFacility(AdminFacility.class);

// Look up the experimenter to sudo for
ExperimenterData sudoUser = admin.lookupExperimenter(ctx, sudoUsername);

// Create a SecurityContext for this user within the user's default group
// and set the 'sudo' flag (i.e. all operations using this context will
// be performed as this user)
SecurityContext sudoCtx = new SecurityContext(sudoUser.getGroupId());
sudoCtx.setExperimenter(sudoUser);
sudoCtx.sudo();

```

(continues on next page)

(continued from previous page)

```
// Get a sudouser's dataset (assume the user has at least one dataset)
BrowseFacility browse = gateway.getFacility(BrowseFacility.class);
Collection<DatasetData> datasets = browse.getDatasets(sudoCtx, sudoUser.getId());
DatasetData sudoDataset = datasets.iterator().next();

// Add a tag to the dataset on behalf of the sudouser (i.e. the sudouser will be
// the owner of tag).
DataManagerFacility dm = gateway.getFacility(DataManagerFacility.class);
TagAnnotationData sudoUserTag = new TagAnnotationData(sudoUsername+"'s tag");
dm.attachAnnotation(sudoCtx, sudoUserTag, sudoDataset);
System.out.println("Added '"+sudoUserTag.getContentAsString()+"' "
    + "to dataset "+sudoDataset.getName()+" on behalf of "+sudoUsername);

// Add a tag to the same dataset as logged in user (i. e. the logged in user will be
// the owner of the tag). Note: This only works in a read-annotate group where the
// logged in user is allowed to annotate the sudouser's data, or the logged in user has
// write permission.
TagAnnotationData adminTag = new TagAnnotationData(user.getUserName()+"'s tag");
// Have to use a SecurityContext for the correct group, otherwise this would fail
// with a security violation
SecurityContext groupContext = new SecurityContext(sudoUser.getGroupId());
dm.attachAnnotation(groupContext, adminTag, sudoDataset);
System.out.println("Added '"+adminTag.getContentAsString()+"' "
    + "to dataset "+sudoDataset.getName()+" as admin.");
```

Further information

For the details behind writing, configuring, and executing a client, please see *Working with OMERO*.

See also:

ZeroC, *OMERO.grid*, *OmeroTools*, *OMERO Application Programming Interface*

3.2.5 OMERO MATLAB language bindings

See *Developing OMERO clients* and *OME-Remote Objects*, for an introduction to **Object**.

Installing the OMERO.matlab toolbox

- Download the latest released version from the [Downloads](#) page.
- Unzip the directory anywhere on your system.
- In MATLAB, move to the newly unzipped directory and run `loadOmero`;
- The MATLAB files are now on your path, and the necessary jars are on your Java classpath. You can change directories and still have access to OMERO.

Once OMERO.matlab is installed, the typical workflow is:

1. *Creating a connection*

2. *Keeping your session alive*
3. *Creating an unencrypted session* (optional)
4. Do some work (load objects, work with them, upload to the server, etc.)
5. *Closing your connection*
6. *Unloading OMERO* (optional)

As a quickstart example, the following lines create a secure connection to a server, read a series of images and close the connection.

```
client = loadOmero(servername);
session = client.createSession(user, password);
client.enableKeepAlive(60);
images = getImages(session, ids);
client.closeSession();
```

Examples of usage of the OMERO.matlab toolbox are provided in the [training examples](#) directory.

Configuring the OMERO.matlab connection

Creating a connection

As described under *Working with OMERO*, there are several ways to configure your connection to an OMERO server. OMERO.matlab comes with a few conveniences for making this work.

If you run `client = loadOmero();` (i.e. `loadOmero` without an input argument), then OMERO.matlab will try to configure the `omero.client` object for you. First, it checks the `ICE_CONFIG` environment variable. If set, it will let the `omero.client` constructor initialize itself. Otherwise, it looks for the file `ice.config` in the current directory. The OMERO.matlab toolbox comes with a default `ice.config` file pointing at `localhost`. To use this configuration file, you should replace `localhost` by your server address.

Alternatively, you can pass the server address to `loadOmero`; to create a client:

```
>> client = loadOmero(servername);
```

Or, if you want a session created directly using the configuration `ice.config` file:

```
>> [client, session] = loadOmero('ice.config');
```

This is equivalent to:

```
>> client = loadOmero(servername, port);
>> session = client.createSession(username, password)
```

where the variables `servername`, `port`, `username` and `password` are the values set in `ice.config` for the previous example. The default port will be used if not specified.

Keeping your session alive

For executing any long running task, you will need a background thread which keeps your session alive. If you are familiar with MATLAB Timers you can use `omeroKeepAlive.m` directly or modify it to your liking. By default the function creates a default 60-second timer.

```
>> [client, session] = loadOmero('ice.config');
>> timer = omeroKeepAlive(client); % Create timer and starts it.
>> ...
>> delete(timer); % Disable the keep-alive
```

Alternatively, you can use the Java-based `enableKeepAlive` method, but it is not configurable from within MATLAB. In that case, you will need to specify the time interval:

```
client.enableKeepAlive(60); % Call session.keepAlive() every 60 seconds
client.closeSession(); % Close session to end the keep-alive
```

Working in a different group

Each session is created within a given context, defining not only the session user but also the session group. The session context can be retrieved using the administration service:

```
eventContext = session.getAdminService().getEventContext();
groupId = eventContext.groupId;
```

Most read and write operations described below are performed in the context of the session group when using the default parameters. Since OMERO 5.1.4, it is possible to specify a different context than the session group for reading and writing data using the `group` parameter/key value in the OMERO.matlab functions. Retrieving objects by identifiers is also done across all groups by default.

See also:

OMERO permissions querying, usage and history

Developer documentation about the OMERO permissions system

Creating an unencrypted session

Once a session has been created, if you want to speed up the data transfer, you can create and use an unencrypted session as:

```
unsecureClient = client.createClient(false);
sessionUnencrypted = unsecureClient.getSession();
```

Closing your connection

When you are done with OMERO, it is critical that you close your connection to save resources:

```
client.closeSession();
clear client;
clear session;
```

If you created an unencrypted session, you will need to close the unsecure session as well:

```
client.closeSession();
unsecureClient.closeSession();
clear client;
clear unsecureClient;
clear session;
clear sessionUnencrypted;
```

Unloading OMERO

Then if you would like, you can unload OMERO as well:

```
unloadOmero();
```

You may see the following warning when unloading OMERO:

```
>> unloadOmero()
Warning: Objects of omero/client class exist - not clearing java
> In javaclasspath>doclear at 377
  In javaclasspath>local_javapath at 194
  In javaclasspath at 105
  In javarmpath at 48
  In unloadOmero at 75
```

```
=====
While unloading OMERO, found java objects left in workspace.
Please remove with 'clear <name>' and then run 'unloadOmero'
again. Printing all objects...
=====
```

Name	Size	Bytes	Class	Attributes
c	1x1		omero.client	

```
Closing session(s) for 1 found client(s): c
```

This means that there is still an OMERO.matlab object in your workspace. If not listed, use `whos` to find such objects, and `clear` to remove them. After that, run `unloadOmero()` again:

```
>> clear c
>> unloadOmero()
```

Warning: You should also unload OMERO before installing a new version of OMERO.matlab or calling `loadOmero` again.

If you need to create another session without unloading/loading OMERO again, use the `omero.client` object directly:

```
>> client = loadOmero(servername,port);
>> client = omero.client(username_1, password_1);
>> session = c.createSession();
```

Reading data

The IContainer service provides methods to load the data management hierarchy in OMERO – projects, datasets, etc.. A list of examples follows indicating how to load projects, datasets, screens.

- **Projects**

The projects owned by the session user in the context of the session group can be retrieved using the `getProjects` function:

```
projects = getProjects(session)
```

If the project identifiers are known, they can be retrieved independently of their owner or group using:

```
projects = getProjects(session, ids)
```

If the projects contain datasets, the datasets will automatically be loaded:

```
for j = 1 : numel(projects) % MATLAB list, index starts at 1
    % Get all the datasets in the Project
    datasetsList = projects(j).linkedDatasetList; % Java List
    % convert it to a MATLAB list for convenience
    datasets = toMatlabList(datasetsList);
    % Iterate through datasets
    for i = 1 : numel(datasets)
        d = datasets(i);
    end
end
```

If the datasets contain images, the images are not automatically loaded. To load the whole graph (projects, datasets, images), pass `true` as an optional argument:

```
% Load the specified Projects and the whole graph
loadedProjects = getProjects(session, ids, true)
% Get the first project
project_1 = loadedProjects(1) % MATLAB array, index starts at 1
% Get all the datasets in the Project
datasets = project_1.linkedDatasetList;
% Get the first dataset in the Java list, index starts at 0
dataset_1 = datasets.get(0);
dataset_name = dataset_1.getName().getValue(); % dataset's name
dataset_id = dataset_1.getId().getValue(); % dataset's id
% Retrieve all the images in the datasets as a Java List (index will start at 0)
imageList = dataset_1.linkedImageList;
% convert it to a MATLAB list for convenience
```

(continues on next page)

(continued from previous page)

```

images = toMatlabList(imageList);
% Iterate through the images
for i = 1 : numel(images)
    image = images(i);
    image_name = image.getName().getValue(); % image's name
    image_id = image.getId().getValue(); % image's id
end

```

Warning: Loading the entire projects/datasets/images graph can be time-consuming and memory-consuming depending on the amount of data.

To return the orphaned datasets i.e. datasets not in a project, as well as the projects, you can query the second output argument of `getProjects`:

```
[projects, orphanedDatasets] = getProjects(session)
```

To filter projects by owner, use the `owner` parameter/key value. A value of `-1` means projects are retrieved independently of their owner:

```

% Returns all projects owned by the specified user in the context of the
% session group
projects = getProjects(session, 'owner', ownerId);
% Returns all projects with the input identifiers owned by the specified
% user
projects = getProjects(session, ids, 'owner', ownerId);
% Returns all projects owned by any user in the context of the session
% group
projects = getProjects(session, 'owner', -1);

```

To filter projects by group, use the `group` parameter/key value. A value of `-1` means projects are retrieved independently of their group:

```

% Returns all projects owned by the session user in the specified group
projects = getProjects(session, 'group', groupId);
% Returns all projects with the input identifiers in the specified group
projects = getProjects(session, ids, 'group', groupId);
% Returns all projects owned by the session user across groups
projects = getProjects(session, 'group', -1);

```

• Datasets

The datasets owned by the session user in the context of the session group can be retrieved using the `getDatasets` function:

```
datasets = getDatasets(session)
```

If the dataset identifiers are known, they can be retrieved independently of their owner or group using:

```
datasets = getDatasets(session, ids)
```

If the datasets contain images, the images are not automatically loaded. To load the whole graph (datasets, images), pass `true` as an optional argument:


```
loadedDatasets = getDatasets(session, ids, true);
% Get the first dataset
dataset_1 = loadedDatasets(1); % MATLAB array, index starts at 1
% Get the all the images in the dataset as the Java list, index starts at 0
imageList = dataset_1.linkedImageList;
```

To filter datasets by owner, use the `owner` parameter/key value. A value of `-1` means datasets are retrieved independently of their owner:

```
% Returns all datasets owned by the specified user in the context of the
% session group
datasets = getDatasets(session, 'owner', ownerId);
% Returns all datasets with the input identifiers owned by the specified
% user
datasets = getDatasets(session, ids, 'owner', ownerId);
% Returns all datasets owned by any user in the context of the session
% group
datasets = getDatasets(session, 'owner', -1);
```

To filter datasets by group, use the `group` parameter/key value. A value of `-1` means datasets are retrieved independently of their group:

```
% Returns all datasets owned by the session user in the specified group
datasets = getDatasets(session, 'group', groupId);
% Returns all datasets with the input identifiers in the specified group
datasets = getDatasets(session, ids, 'group', groupId);
% Returns all datasets owned by the session user across groups
datasets = getDatasets(session, 'group', -1);
```

• Images

The images owned by the session user in the context of the session group can be retrieved using the `getImages` function:

```
images = getImages(session)
```

If the image identifiers are known, they can be retrieved independently of their owner or group using:

```
images = getImages(session, ids)
```

All the images contained in a subset of datasets of known identifiers `datasetsIds` can be returned independently of their owner or group using:

```
datasetImages = getImages(session, 'dataset', datasetsIds)
```

All the images contained in all the datasets under a subset of projects of known identifiers `projectIds` can be returned independently of their owner or group using:

```
projectImages = getImages(session, 'project', projectIds)
```

To filter images by owner, use the `owner` parameter/key value. A value of `-1` means images are retrieved independently of their owner:

```
% Returns all images owned by the specified user in the context of the
% session group
images = getImages(session, 'owner', ownerId);
```

(continues on next page)

(continued from previous page)

```
% Returns all images with the input identifiers owned by the specified user
images = getImages(session, ids, 'owner', ownerId);
% Returns all images owned by any user in the context of the session
% group
images = getImages(session, 'owner', -1);
```

To filter images by group, use the group parameter/key value. A value of -1 means images are retrieved independently of their group:

```
% Returns all images owned by the session user in the specified group
images = getImages(session, 'group', groupId);
% Returns all images with the input identifiers in the specified group
images = getImages(session, ids, 'group', groupId);
% Returns all images owned by the session user across groups
images = getImages(session, 'group', -1);
```

The Image-Pixels model (see *OME-Remote Objects*) implies you need to use the Pixels objects to access valuable data about the Image:

```
pixels = image.getPrimaryPixels();
sizeZ = pixels.getSizeZ().getValue(); % The number of z-sections.
sizeT = pixels.getSizeT().getValue(); % The number of timepoints.
sizeC = pixels.getSizeC().getValue(); % The number of channels.
sizeX = pixels.getSizeX().getValue(); % The number of pixels along the X-axis.
sizeY = pixels.getSizeY().getValue(); % The number of pixels along the Y-axis.
```

- **Screens**

The screens owned by the session user in the context of the session group can be retrieved using the `getScreens` function:

```
screens = getScreens(session)
```

If the screen identifiers are known, they can be retrieved independently of their owner or group using:

```
screens = getScreens(session, ids)
```

Note that the wells are not loaded. The plate objects can be accessed using:

```
for j = 1 : numel(screens), % MATLAB array, index start at 1
    platesList = screens(j).linkedPlateList; % Java List, index start at 0
    for i = 0 : platesList.size()-1,
        plate = platesList.get(i);
        plateAcquisitionList = plate.copyPlateAcquisitions(); % Java List
        for k = 0 : plateAcquisitionList.size()-1,
            pa = plateAcquisitionList.get(i);
        end
    end
```

To return the orphaned plates as well as the screens, you can query the second output argument of `getScreens`:

```
[screens, orphanedPlates] = getScreens(session)
```

To filter screens by owner, use the owner parameter/key value. A value of -1 means screens are retrieved independently of their owner:

```
% Returns all screens owned by the specified user in the context of the
% session group
screens = getScreens(session, 'owner', ownerId);
% Returns all screens with the input identifiers owned by the specified
% user
screens = getScreens(session, ids, 'owner', ownerId);
% Returns all screens owned by any user in the context of the session
% group
screens = getScreens(session, 'owner', -1);
```

To filter screens by group, use the `group` parameter/key value. A value of `-1` means screens are retrieved independently of their group:

```
% Returns all screens owned by the session user in the specified group
screens = getScreens(session, 'group', groupId);
% Returns all screens with the input identifiers in the specified group
screens = getScreens(session, ids, 'group', groupId);
% Returns all screens owned by the session user across groups
screens = getScreens(session, 'group', -1);
```

• Plates

The screens owned by the session user in the context of the session group can be retrieved using the `getPlates` function:

```
plates = getPlates(session)
```

If the plate identifiers are known, they can be retrieved independently of their owner or group using:

```
plates = getPlates(session, ids)
```

To filter plates by owner, use the `owner` parameter/key value. A value of `-1` means plates are retrieved independently of their owner:

```
% Returns all plates owned by the specified user in the context of the
% session group
plates = getPlates(session, 'owner', ownerId);
% Returns all plates with the input identifiers owned by the specified user
plates = getPlates(session, ids, 'owner', ownerId);
% Returns all plates owned by any user in the context of the session
% group
plates = getPlates(session, 'owner', -1);
```

To filter plates by group, use the `group` parameter/key value. A value of `-1` means plates are retrieved independently of their group:

```
% Returns all plates owned by the session user in the specified group
plates = getPlates(session, 'group', groupId);
% Returns all plates with the input identifiers in the specified group
plates = getPlates(session, ids, 'group', groupId);
% Returns all plates owned by the session user across groups
plates = getPlates(session, 'group', -1);
```

• Wells

Given a plate identifier, the wells can be loaded using the `findAllByQuery` method:

```

wellList = session.getQueryService().findAllByQuery(
['select well from Well as well '...
'left outer join fetch well.plate as pt '...
'left outer join fetch well.wellSamples as ws '...
'left outer join fetch ws.plateAcquisition as pa '...
'left outer join fetch ws.image as img '...
'left outer join fetch img.pixels as pix '...
'left outer join fetch pix.pixelsType as pt '...
'where well.plate.id = ', num2str(plateId)], []);
% wellList is a Java List, index starts at 0
for j = 0 : wellList.size()-1,
    well = wellList.get(j);
    wellsSampleList = well.copyWellSamples();
    well.getId().getValue()
    % The wellList returned from the server is not sorted by wellIds,
    % please extract the wellRow and wellColumn for every well,
    % to populate your results appropriately
    wellRow = well.getRow().getValue();
    wellColumn = well.getColumn().getValue();
    for i = 0 : wellsSampleList.size()-1,
        ws = wellsSampleList.get(i);
        ws.getId().getValue()
        pa = ws.getPlateAcquisition();
    end
end

```

- **Channel**

A channel associated to an image has an object called a logicalChannel associated to it. That entity contains valuable information e.g. emission wavelength, name, etc. Given an Image, retrieve channels associated to an image on the OMERO server and the name of the channel:

```

channels = loadChannels(session, image);
for j = 1 : numel(channels) % MATLAB array
    channel = channels(j);
    channelId = channel.getId().getValue();
    lc = channel.getLogicalChannel();
    channelName = lc.getName().getValue();
end

```

Raw data access

You can retrieve data, plane by plane or retrieve a stack. The values are z in $[0, \text{sizeZ} - 1]$, c in $[0, \text{sizeC} - 1]$ and t in $[0, \text{sizeT} - 1]$.

- **Plane**

The plane of an input image at coordinates (z , c , t) can be retrieved using the `getPlane` function:

```
plane = getPlane(session, image, z, c, t);
```

Alternatively, the image identifier can be passed to the function:

```
plane = getPlane(session, imageId, z, c, t);
```

- **Tile**

The tile of an input image at coordinates (z, c, t) originated at (x, y) (where x in [0, sizeX - 1], y in [0, sizeY - 1]) and of dimensions (w, h) can be retrieved using the `getTile` function:

```
tile = getTile(session, image, z, c, t, x, y, w, h);
```

Alternatively, the image identifier can be passed to the function:

```
tile = getTile(session, imageId, z, c, t, x, y, w, h);
```

- **Stack**

The stack of an input image at coordinates (c, t) can be retrieved using the `getStack` function:

```
stack = getStack(session, image, c, t);
```

Alternatively, the image identifier can be passed to the function:

```
stack = getStack(session, imageId, c, t);
```

All the methods described above will internally initialize a raw pixels store to retrieve the pixels data and close this store at the end of the call. This is inefficient when multiple planes/tiles/stacks need to be retrieved. For each function, it is possible to initialize a pixels store and pass this store directly to the pixel retrieval function, e.g.:

```
[store, pixels] = getRawPixelsStore(session, image);
for z = 0 : sizeZ - 1
    for c = 0 : sizeC - 1
        for t = 0 : sizeT - 1
            plane = getPlane(pixels, store, z, c, t);
        end
    end
end
store.close();
```

- **Hypercube**

This is useful when you need the Pixels intensity.

```
% Create the store to load the stack. No access via the gateway
store = session.createRawPixelsStore();
% Indicate the pixels set you are working on
store.setPixelsId(pixelsId, false);

% Offset values in each dimension XYZCT
offset = java.util.ArrayList;
offset.add(java.lang.Integer(0));
offset.add(java.lang.Integer(0));
offset.add(java.lang.Integer(0));
offset.add(java.lang.Integer(0));
offset.add(java.lang.Integer(0));

size = java.util.ArrayList;
```

(continues on next page)

(continued from previous page)

```

size.add(java.lang.Integer(sizeX));
size.add(java.lang.Integer(sizeY));
size.add(java.lang.Integer(sizeZ));
size.add(java.lang.Integer(sizeC));
size.add(java.lang.Integer(sizeT));

% Indicate the step in each direction,
% step = 1, will return values at index 0, 1, 2.
% step = 2, values at index 0, 2, 4, etc.
step = java.util.ArrayList;
step.add(java.lang.Integer(1));
step.add(java.lang.Integer(1));
step.add(java.lang.Integer(1));
step.add(java.lang.Integer(1));
step.add(java.lang.Integer(1));
% Retrieve the data
store.getHypercube(offset, size, step);
% Close the store
store.close();

```

See also:**RawDataAccess.m**

Example script showing methods to retrieve the pixel data from an image

Annotations

- **Reading annotations by ID**

If the identifier of the annotation of a given type is known, the annotation can be retrieved from the server using the generic `getAnnotations` function:

```
tagAnnotations = getAnnotations(session, 'tag', tagIds);
```

Shortcut functions are available for the main object and annotation types, e.g. to retrieve tag annotations:

```
tagAnnotations = getTagAnnotations(session, tagIds);
```

- **Reading annotations linked to an object**

The annotations of a given type linked to a given object can be retrieved using the generic `getObjectAnnotations` function:

```
tagAnnotations = getObjectAnnotations(session, 'tag', 'image', imageIds);
```

Shortcut functions are available for the main object and annotation types, e.g. to retrieve the tag annotations linked to images:

```
tagAnnotations = getImageTagAnnotations(session, imageIds);
```

Annotations can be filtered by namespace. To include only annotations with a given namespace `ns`, use the `include` parameter/key value:

```
tagAnnotations = getImageTagAnnotations(session, imageIds, 'include', ns);
```

To exclude all annotations with a given namespace ns, use the `exclude` parameter/key value:

```
tagAnnotations = getImageTagAnnotations(session, imageIds, 'exclude', ns);
```

By default, only the annotations owned by the session owner are returned. To specify the owner of the annotations, use the `owner` parameter/key value pair. For instance to return all tag annotations owned by user with an identifier equals to 5:

```
tagAnnotations = getImageTagAnnotations(session, imageIds, 'owner', 5);
```

To retrieve all annotations independently of their owner, use `-1` as the owner identifier:

```
tagAnnotations = getImageTagAnnotations(session, imageIds, 'owner', -1);
```

• Reading file annotations

The content of a file annotation can be downloaded to local disk using the `getFileAnnotationContent` function. If the file annotation has been retrieved from the server as `fileAnnotation`, then the content of its `OriginalFile` can be downloaded under `target_file` using:

```
getFileAnnotationContent(session, fileAnnotation, target_file);
```

Alternatively, if only the identifier of the file annotation `faId` is known:

```
getFileAnnotationContent(session, faId, target_file);
```

• Writing and linking annotations

New annotations can be created using the corresponding `write*Annotation` function:

```
% Create a comment annotation
commentAnnotation = writeCommentAnnotation(session, 'comment');
% Create a double annotation
doubleAnnotation = writeDoubleAnnotation(session, .5);
% Create a map annotation
mapAnnotation = writeMapAnnotation(session, 'key', value);
% Create a tag annotation
tagAnnotation = writeTagAnnotation(session, 'tag name');
% Create a timestamp annotation
timestampAnnotation = writeTimestampAnnotation(session, now);
% Create an XML annotation
xmlAnnotation = writeXmlAnnotation(session, xmlString);
```

File annotations can also be created from the content of a `local_file_path`:

```
fileAnnotation = writeFileAnnotation(session, local_file_path);
```

Each annotation creation function uses the context of the session group by default. To create the annotation in a different group, use the `group` key/value pair:

```
commentAnnotation = writeCommentAnnotation(session, 'comment', 'group', groupId);
doubleAnnotation = writeDoubleAnnotation(session, .5, 'group', groupId);
mapAnnotation = writeMapAnnotation(session, 'key', value, 'group', groupId);
```

(continues on next page)

(continued from previous page)

```

tagAnnotation = writeTagAnnotation(session, 'tag name', 'group', groupId);
timestampAnnotation = writeTimestampAnnotation(session, now, 'group', groupId);
xmlAnnotation = writeXmlAnnotation(session, xmlString, 'group', groupId);
fileAnnotation = writeFileAnnotation(session, local_file_path, 'group', groupId);

```

Existing annotations can be linked to existing objects on the server using the `linkAnnotation` function. For example, to link a tag annotation and a file annotation to the image `image_id`:

```

link1 = linkAnnotation(session, tagAnnotation, 'image', imageId);
link2 = linkAnnotation(session, fileAnnotation, 'image', imageId);

```

For existing file annotations, it is possible to replace the content of the original file without having to recreate a new file annotation using the `updateFileAnnotation` function. If the file annotation has been retrieved from the server as `fileAnnotation`, then the content of its `OriginalFile` can be replaced by the content of `local_file_path` using:

```
updateFileAnnotation(session, fileAnnotation, local_file_path);
```

See also:

WriteData.m

Example script showing methods to write, link and retrieve annotations.

Writing data

• Projects/Datasets

Projects and datasets can be created in the context of the session group using the `createProject` and `createDataset` functions:

```

% Create a new project in the context of the session group
newproject = createProject(session, 'project name');
% Create a new dataset in the context of the session group
newdataset = createDataset(session, 'dataset name');

```

Writing projects/datasets in a different context than the session group can be achieved by passing the group identifier using the `group` parameter:

```

% Create a new project in the specified group
newproject = createProject(session, 'project name', 'group', groupId);
% Create a new dataset in the specified group
newdataset = createDataset(session, 'dataset name', 'group', groupId);

```

When creating a dataset, it is possible to link it to an existing project using either the project object or its identifier. In this case, the group context is determined by the parent project:

```

% Create two new projects in different groups
project1 = createProject(session, 'project name');
project2 = createProject(session, 'project name', 'group', groupId);
% Create new datasets linked to each project
dataset1 = createDataset(session, 'dataset name', project1);
dataset2 = createDataset(session, 'dataset name', project2.getId().getValue());

```

• Screens/Plates

Screens and plates can be created in the context of the session group using the `createScreen` and `createPlate` functions:


```
% Create a new screen in the context of the session group
newscreen = createScreen(session, 'screen name');
% Create a new plate in the context of the session group
newplate = createPlate(session, 'plate name');
```

Writing screens/plates in a different context than the session group can be achieved by passing the group identifier using the *group* parameter:

```
% Create a new screen in the specified group
newscreen = createScreen(session, 'screen name', 'group', groupId);
% Create a new plate in the specified group
newplate = createPlate(session, 'plate name', 'group', groupId);
```

When creating a plate, it is possible to link it to an existing screen using either the screen object or its identifier. In this case, the group context is determined by the parent screen:

```
% Create two new projects in different groups
screen1 = createScreen(session, 'screen name');
screen2 = createScreen(session, 'screen name', 'group', groupId);
% Create new datasets linked to each project
plate1 = createPlate(session, 'plate name', screen1);
plate2 = createPlate(session, 'plate name', screen2.getId().getValue());
```

See also:

WriteData.m

Example script showing methods to create projects, datasets, plates and screens.

How to use OMERO tables

- **Create a table.** In the following example, a table is created with 2 columns and is linked to an Image.

```
name = char(java.util.UUID.randomUUID());
columns = javaArray('omero.grid.Column', 2);
columns(1) = omero.grid.LongColumn('Uid', 'testLong', []);
valuesString = javaArray('java.lang.String', 1);
columns(2) = omero.grid.StringColumn('MyStringColumn', '', 64, valuesString);

% Create a new table.
table = session.sharedResources().newTable(1, name);

% Initialize the table
table.initialize(columns);

% Create and populate omero.grid (The following java wrapping logic is compatible ↵
↵ Matlab2014b onwards)
data = javaArray('omero.grid.Column', 2);
data(1) = omero.grid.LongColumn('Uid', 'test Long', [2]);
valuesString = javaArray('java.lang.String', 1);
valuesString(1) = java.lang.String('add');
data(2) = omero.grid.StringColumn('MyStringColumn', '', 64, valuesString);

% Add data to the table.
```

(continues on next page)

(continued from previous page)

```

table.addData(data);
file = table.getOriginalFile(); % if you need to interact with the table

% link table to an Image
fa = omero.model.FileAnnotationI;
fa.setFile(file);
% Currently OMERO.tables are displayed only in OMERO.web and
% for Screen/plate/wells alone. In all cases the file annotation
% needs to contain a namespace.
fa.setNs(rstring(omero.constants.namespaces.NSBULKANNOTATIONS.value));
link = linkAnnotation(session, fa, 'image', imageId);

```

- **Read the contents of the table.**

```

of = omero.model.OriginalFileI(file.getId(), false);
tablePrx = session.sharedResources().openTable(of);

% Read headers
headers = tablePrx.getHeaders();
for i = 1 : size(headers, 1)
    headers(i).name; % name of the header
    % Do something
end

% Depending on the size of table, you may only want to read some blocks.
cols = [0:size(headers, 1)-1]; % The number of columns you wish to read.
rows = [0:tablePrx.getNumberOfRows()-1]; % The number of rows you wish to read.
data = tablePrx.slice(cols, rows); % Read the data.
c = data.columns;
for i = 1 : size(c)
    column = c(i);
    % Do something
end
tablePrx.close(); % Important to close when done.

```

ROIs

To learn about the model, see the [developers guide to the ROI model](#). Note that annotations can be linked to ROI.

- **Creating ROI**

This example creates a ROI with shapes, a rectangle, an ellipse and a polygon, and attaches it to an image:

```

% First create a rectangular shape.
rectangle = createRectangle(0, 0, 10, 20);
% Indicate on which plane (z, c, t) to attach the shape
setShapeCoordinates(rectangle, 0, 0, 0);

% First create an ellipse shape.
ellipse = createEllipse(0, 0, 10, 20);
% Indicate on which plane (z, c, t) to attach the shape
setShapeCoordinates(ellipse, 0, 0, 0);

```

(continues on next page)

(continued from previous page)

```

% First create a polygon shape.
% Specify x-coordinates, y-coordinates
polygon = createPolygon([1 5 10 8], [1 5 5 10]);
% Indicate on which plane (z, c, t) to attach the shape
setShapeCoordinates(polygon, 0, 0, 0);

% Create the roi.
roi = omero.model.RoiI;
% Attach the shapes to the roi, several shapes can be added.
roi.addShape(rectangle);
roi.addShape(ellipse);
roi.addShape(polygon);

% Link the roi and the image
roi.setImage(omero.model.ImageI(imageId, false));
% Save
iUpdate = session.getUpdateService();
roi = iUpdate.saveAndReturnObject(roi);
% Check that the shape has been added.
numShapes = roi.sizeOfShapes;
for ns = 1 : numShapes
    shape = roi.getShape(ns-1);
end

```

See also:

ROI utility functions

OMERO.matlab functions for creating and managing Shape and ROI objects.

- Retrieving ROIs linked to an image

```

service = session.getRoiService();
roiResult = service.findByImage(imageId, []);
rois = roiResult.rois;
n = rois.size;
shapeType = '';
for thisROI = 1 : n
    roi = rois.get(thisROI-1);
    numShapes = roi.sizeOfShapes;
    for ns = 1 : numShapes
        shape = roi.getShape(ns-1);
        if (isa(shape, 'omero.model.Rectangle'))
            rectangle = shape;
            rectangle.getX().getValue();
        elseif (isa(shape, 'omero.model.Ellipse'))
            ellipse = shape;
            ellipse.getX().getValue();
        elseif (isa(shape, 'omero.model.Point'))
            point = shape;
            point.getX().getValue();
        elseif (isa(shape, 'omero.model.Line'))
            line = shape;

```

(continues on next page)

(continued from previous page)

```

        line.getX1().getValue();
    end
end
end

```

- Adding Transforms to a Shape object

```

% Apply rotation alone to an ellipse object
% (angle of rotation set to 10 degrees)
% create ellipse (shape object)
ellipse = createEllipse(0, 0, 10, 20);
setShapeCoordinates(ellipse, 0, 0, 0);
% set angle of rotation
theta = 10;
% create transform object
newTform = omero.model.AffineTransformI;
newTform.setA00(rdouble(cos(theta)));
newTform.setA10(rdouble(-sin(theta)));
newTform.setA01(rdouble(sin(theta)));
newTform.setA11(rdouble(cos(theta)));
newTform.setA02(rdouble(0));
newTform.setA12(rdouble(0));
% apply transform
ellipse.setTransform(newTform);
% Create the ROI
roi = omero.model.RoiI;
roi.addShape(ellipse);
roi = session.getUpdateService().saveAndReturnObject(roi);

```

- Retrieving Transforms linked to an Image

```

for i = 1 : nShapes
    shape = roi.getShape(i - 1);

    %http://blog.openmicroscopy.org/data-model/future-plans/2016/06/20/shape-transforms/
    transform = shape.getTransform();
    xScaling = transform.getA00().getValue();
    xShearing = transform.getA01().getValue();
    xTranslation = transform.getA02().getValue();

    yScaling = transform.getA11().getValue();
    yShearing = transform.getA10().getValue();
    yTranslation = transform.getA12().getValue();

    %tformMatrix = [A00, A10, 0; A01, A11, 0; A02, A12, 1];
    tformMatrix = [xScaling, yShearing, 0; xShearing, yScaling, 0; xTranslation,
    ↪yTranslation, 1];

    fprintf(1, 'Shape Type : %s\n', char(shape.toString));
    fprintf(1, 'xScaling : %s\n', num2str(tformMatrix(1,1)));
    fprintf(1, 'yScaling : %s\n', num2str(tformMatrix(2,2)));
    fprintf(1, 'xShearing : %s\n', num2str(tformMatrix(2,1)));

```

(continues on next page)

(continued from previous page)

```

fprintf(1, 'yShearing : %s\n', num2str(tformMatrix(1,2)));
fprintf(1, 'xTranslation: %s\n', num2str(tformMatrix(3,1)));
fprintf(1, 'yTranslation: %s\n', num2str(tformMatrix(3,2)));
end

```

- Removing a shape from ROI

```

// Retrieve the roi linked to an image
service = session.getRoiService();
roiResult = service.findByImage(imageId, []);
n = rois.size;
for thisROI = 1 : n
    roi = rois.get(thisROI-1);
    numShapes = roi.sizeOfShapes;
    for ns = 1:numShapes
        shape = roi.getShape(ns-1);
        % Remove the shape
        roi.removeShape(shape);
    end
    % Update the roi.
    roi = iUpdate.saveAndReturnObject(roi);
end

```

- Analyzing shapes

```

// Retrieve the roi linked to an image
service = session.getRoiService();
roiResult = service.findByImage(imageId, []);
n = rois.size;
toAnalyse = java.util.ArrayList;
for thisROI = 1 : n
    roi = rois.get(thisROI-1);
    numShapes = roi.sizeOfShapes;
    for ns = 1:numShapes
        shape = roi.getShape(ns-1);
        toAnalyse.add(java.lang.Long(shape.getId().getValue()));
    end
end
//For convenience, we assume the shapes are on the first plane
z = 0;
c = 0;
t = 0;
stats = service.getShapeStatsRestricted(toAnalyse, z, t, [c]);
calculated = stats(1,1);
mean = calculated.mean(1,1);

```

Deleting data

It is possible to delete projects, datasets, images, ROIs, etc. and objects linked to them depending on the specified options (see *Deleting in OMERO*). For example, images of known identifiers can be deleted from the server using the `deleteImages` function:

```
deleteImages(session, imageIds);
```

See also:

`deleteProjects`, `deleteDatasets`, `deleteScreens`, `deletePlates`

Utility functions to delete objects.

Rendering images

The `RenderImages.m` example script shows how to initialize the rendering engine and render an image.

Creating Image

The `CreateImage.m` example script shows how to create an image in OMERO. A similar approach can be applied when uploading an image. To upload individual planes onto the server, the data must be converted into a byte (int8) array first. If the `Pixels` object has been created, this conversion can be done using the `toByteArray` function.

3.2.6 OMERO C++ language bindings

Using the `Ice C++ language mapping` from `ZeroC`, OMERO provides native access to your data from C++ code. `CMake` is used for building the C++ bindings.

Binaries are not provided, therefore it will be necessary for you to compile your own.

Prerequisites

- The OMERO source code
- A C++ compiler
 - GCC is recommended for Linux and MacOS X
 - **Visual Studio** or the **Platform SDK** for Windows
- The ZeroC **Ice** libraries, headers and slice definitions
- **cmake**
- Google Test (optional; needed to build the unit and integration tests)

Note: Users of **Visual Studio** with **Ice** 3.6 will encounter this error while building `OmeroCpp`: `LINK : fatal error LNK1189: library limit of 65535 objects exceeded` and will be unable to build *OMERO C++ language bindings* for Windows as a result of a 16-bit limitation in the Windows PE-COFF executable format used for DLLs, even on 64-bit systems. It is hoped that **Ice** 3.7 will resolve the problem since it generates far fewer symbols than 3.6.

Restrictions

If you are restricted to a specific version of GCC or Ice, you may need to obtain or build a compatible version of Ice or GCC, respectively.

Preparing to build

Begin by following the instructions under *Installing OMERO from source* to acquire the source code. Be **sure** that the git branch you are using matches the version of your server!

The location of your Ice installation should be automatically detected if installed into a standard location. If this is not the case, set the location of your Ice installation using the ICE_HOME environment variable or the `cmake -DICE_HOME` or `cmake -DICE_SLICE_DIR` **cmake** options for your Ice installation (see below). Some possible locations for the 3.6.5 version of Ice follow. Note these are just examples; you need to adjust them for the Ice installation path and version in use on your system.

- Ice built from source and installed into /opt:
`export ICE_HOME=/opt/Ice-3.6.5`
- Ice installed on Linux using RPM packages:
`export ICE_HOME=/usr/share/Ice-3.6.5`
- MacOS X with homebrew:
`export ICE_HOME=/usr/local/Cellar/ice/3.6.5`
- Windows using **Visual Studio**:
`set ICE_HOME=C:\Program Files (x86)\ZeroC\Ice-3.6.5`

Note: If the Ice headers and libraries are not automatically discovered, these will need to be specified using appropriate **cmake** options (see below).

Building the library

The shared library and examples are always built by default. The unit and integration tests are built if Google test (gtest) is detected.

On Linux, Unix or MacOS X with **make**:

```
export GTEST_ROOT=/path/to/gtest
mkdir omero-build
cd omero-build
cmake [-Dtest=(TRUE|FALSE)] [cmake options] /path/to/openmicroscopy
make
```

For example:

```
cmake "-DCMAKE_CXX_FLAGS=$CMAKE_CXX_FLAGS" \
"-DCMAKE_EXE_LINKER_FLAGS=$CMAKE_LD_FLAGS" \
"-DCMAKE_MODULE_LINKER_FLAGS=$CMAKE_LD_FLAGS" \
"-DCMAKE_SHARED_LINKER_FLAGS=$CMAKE_LD_FLAGS" \
-DCMAKE_VERBOSE_MAKEFILE:BOOL=ON /path/to/openmicroscopy
make -j8
```

If you would like to build the C++ tests, run the above with the `GTEST_ROOT` environment variable set.

Note: When **cmake** is run, it will run `./build.py build-default` in the openmicroscopy source tree to generate some of the C++ and Ice sources. If you have previously done a build by running `./build.py`, this step will be skipped. However, if you have recently switched branches *without cleaning the source tree*, please run `./build.py clean` in the source tree to clean up all the generated files prior to running **cmake**.

If the build fails with errors such as

```
/usr/include/Ice/ProxyHandle.h:176:13: error: 'upCast' was not declared in this scope,
and no declarations were found by argument-dependent lookup at the point of
instantiation
```

this is caused by the Ice headers being buggy, and newer versions of GCC rejecting the invalid code. To compile in this situation, add `-fpermissive` to `CXXFLAGS` to allow the invalid code to be accepted, but do note that this may also mask other problems so should not be used unless strictly needed.

cmake build configuration

cmake supports configuration of the build using many different environment variables and options; for a full list, see the [cmake reference documentation](#). The following environment variables are commonly needed:

CMAKE_INCLUDE_PATH

Directories to be searched for include files, for example

`/opt/Ice-3.6.5/include`

A `:` or `;` separator character is used to separate directories, depending on the platform. Note these are used only for feature tests, not for passing to the compiler when building, for which `CMAKE_CXX_FLAGS` is needed.

CMAKE_LIBRARY_PATH

Directories to be searched for libraries, for example

`/opt/Ice-3.6.5/lib`

Directories are separated by `:` or `;` as with `CMAKE_INCLUDE_PATH`. Note these are used only for feature tests and finding libraries, not for passing to the linker when building, for which `CMAKE_*_LINKER_FLAGS` is needed.

CXX

C++ compiler executable. Useful with [ccache](#).

CXXFLAGS

C++ compiler flags. Use of `CMAKE_CXX_FLAGS` is preferred.

ICE_HOME

The location of the Ice installation. If this is not sufficient to discover the correct binary and library directories, they may otherwise be manually specified with the options below. Likewise for the `include` and `slice` directories. This may also be set as a **cmake** cache variable (see below).

VERBOSE

If set to `1`, show the actual build commands rather than the pretty “Compiling XYZ...” statements.

In addition, **cmake** options may be defined directly when running **cmake**. Commonly needed options include:

-DCMAKE_PREFIX_PATH

Search this location when searching for programs, headers and libraries. Use to search `/usr/local` or `/opt/Ice`, for example. More specific search locations may be specified using `cmake -DCMAKE_INCLUDE_PATH`, `cmake -DCMAKE_LIBRARY_PATH` and `cmake -DCMAKE_PROGRAM_PATH` separately, if required.

-DCMAKE_INCLUDE_PATH

Search this location when searching for headers. Use to include `/usr/local/include` or `/opt/Ice/include`, for example.

-DCMAKE_LIBRARY_PATH

Search this location when searching for libraries. Use to include `/usr/local/lib` or `/opt/Ice/lib`, for example.

-DCMAKE_PROGRAM_PATH

Search this location when searching for programs. Use to include `/usr/local/bin` or `/opt/Ice/bin`, for example.

-DCMAKE_CXX_FLAGS

C++ compiler flags. Use to set any additional linker flags desired.

-DCMAKE_EXE_LINKER_FLAGS

Executable linker flags. Use to set any additional linker flags desired.

-DCMAKE_MODULE_LINKER_FLAGS

Loadable module linker flags. Use to set any additional linker flags desired.

-DCMAKE_SHARED_LINKER_FLAGS

Shared library linker flags. Use to set any additional linker flags desired.

-DCMAKE_VERBOSE_MAKEFILE

Default to printing all commands executed by make. This may be overridden with the make `VERBOSE` variable.

-DIce_HOME

The location of the Ice installation. If this is not sufficient to discover the correct binary and library directories, they may otherwise be manually specified with the options below. Likewise for the `include` and `slice` directories.

-DIce_SLICE2XXX_EXECUTABLE

Specific location of individual Ice `slice2xxx` programs, e.g. `Ice_SLICE2CPP_EXECUTABLE` for **slice2cpp** or `Ice_SLICE2JAVA_EXECUTABLE` for **slice2java**. These are typically found in `${ICE_HOME}/bin` or on the default PATH. These will not normally require setting.

-DIce_INCLUDE_DIR

Location of Ice headers. This is typically `${ICE_HOME}/include` or on the default include search path. This will not normally require setting.

-DIce_SLICE_DIR

Location of Ice slice interface definitions. This is typically `${ICE_HOME}/slice`. Use for installations where `cmake -DIce_HOME` does not contain `slice` or situations where you wish to build without setting `cmake -DIce_HOME`. Note that when building using **build.py**, rather than building directly with **cmake**, the `SLICEPATH` environment variable should be used instead (the **ant** build can't use the **cmake** variables since it only runs **cmake** after a full build of the Java server).

-DIce_<C>_LIBRARIES

Specific libraries for Ice component `<C>`, where `<C>` is the uppercased name of the Ice component, e.g. `ICE` for the Ice component, `ICEUTIL` for the IceUtil component or `GLACIER2` for the Glacier2 component. These libraries are typically found in `${ICE_HOME}/lib` or on the default library search path. These will not normally require setting.

-DIce_DEBUG

Set to ON to print detailed diagnostics about the detected Ice installation. Use if there are any problems finding Ice.

cmake offers many additional options. Please refer to the [documentation](#) for further details, in particular to the [variables which change the behavior](#) of the build.

Visual Studio configuration

Warning: OMERO.cpp will not currently build on Windows due to exceeding DLL symbol limits on this platform, leading to a failure when linking the DLL. It is hoped that this platform limitation can be worked around in a future OMERO release.

cmake has full support for **Visual Studio**. Use the **cmake** `-G` option to set the generator for your Visual Studio version, with a `Win64` suffix for an x64 build. The correct Ice programs and libraries for your Ice installation should be automatically discovered.

```
cmake -G "Visual Studio 11 Win64" [cmake options] /path/to/openmicroscopy
```

This is for a 64-bit **Visual Studio 2012** build. Modify appropriately for other versions and compilers. Running

```
cmake --help
```

will list the available generators for your platform (without the `Win64` suffix).

Once **cmake** has finished running, the generated project and solution files may be then opened in **Visual Studio**, or built directly using the **msbuild** command-line tool (make sure that the **Visual Studio** command prompt matches the generator chosen) or by running:

```
cmake --build .
```

As for the Unix build, above, it is also possible to build on Windows using **build.py** or **ant**, providing that you configure the generator appropriately using the correct **cmake** options. However, this will not work for all generators reliably, and the Windows shell quoting makes passing nested quotes to ant quite tricky, so running **cmake** by hand is recommended.

Note: It may be necessary to specify `/Zm1000` as an additional compiler setting.

Installing the library

If using **make**, run:

```
make [DESTDIR=/path/to/staging/directory] install
```

If using another build system, please invoke the equivalent install target for that system.

Using the library

To use *OMERO C++ language bindings* it is necessary to point your compiler and linker at the mentioned directories above. A simple GNU **make** Makefile might look like this:

```

1 #
2 # MAKEFILE:
3 #
4 # Where the OMERO distribution was installed
5 OMERO_DIST ?= /opt/omero
6
7 # Where the Ice lib/ and include/ directories are to be found
8 ICE_HOME ?= /usr/share/Ice
9
10 INCLUDES=-I$(OMERO_DIST)/include -I$(ICE_HOME)/include
11
12 LIBS = -L$(OMERO_DIST)/lib -L$(ICE_HOME)/lib -L$(ICE_HOME)/lib64 \
13        -lIce -lIceUtil -lGlacier2 -lomero-client
14
15 LIBPATH = $(LD_LIBRARY_PATH):$(ICE_HOME)/lib:$(ICE_HOME)/lib64:$(OMERO_DIST)/lib
16
17 .PHONY: clean run
18
19 yourcode.o: yourcode.cpp
20     $(CXX) $(CXXFLAGS) -c -o $@ $< $(INCLUDES)
21
22 yourcode: yourcode.o
23     $(CXX) -o $@ $^ $(LIBS)
24
25 run: yourcode
26     LD_LIBRARY_PATH="$(LIBPATH)" ./yourcode --Ice.Config=../etc/ice.config
27
28 clean:
29     rm -f yourcode *.o *~ core

```

A trivial example: yourcode.cpp

A simple example might look something like the following:

```

1 //
2 // yourcode.cpp:
3 //
4
5 // Domain
6 #include <omero/client.h>
7 #include <omero/api/IAdmin.h>
8 // Std
9 #include <iostream>
10 #include <cassert>
11 #include <vector>
12 #include <time.h>
13 #include <map>

```

(continues on next page)

(continued from previous page)

```

14
15 using namespace std;
16
17 /*
18  * Pass "--Ice.Config=your_config_file" to the executable, or
19  * set the ICE_CONFIG environment variable.
20  */
21 int main(int argc, char* argv[])
22 {
23     omero::client_ptr omero = new omero::client(argc, argv);
24     omero::api::ServiceFactoryPrx sf = omero->createSession();
25     sf->closeOnDestroy();
26
27     // IAdmin is responsible for all user/group creation, password changing, etc.
28     omero::api::IAdminPrx admin = sf->getAdminService();
29
30     // Who you are logged in as.
31     cout << admin->getEventContext()->userName << endl;
32
33     // These two services are used for database access
34     omero::api::IQueryPrx query = sf->getQueryService();
35     omero::api::IUpdatePrx update = sf->getUpdateService();
36
37     return 0;
38 }

```

This code does not do much. It creates a server session, loads a few services, and prints the user's name. For serious examples, see [Working with OMERO](#).

Compiling and running your code

To compile and run **yourcode**, download the two files above (Makefile and yourcode.cpp) and then in a shell:

```
make OMERO_DIST=dist yourcode
LD_LIBRARY_PATH=dist/lib ./yourcode --Ice.Config=dist/etc/ice.config
```

where you have edited dist/etc/ice.config to contain the values:

```
omero.host=localhost
omero.user=your_name
omero.pass=your_password
```

Alternatively, you can pass these on the command-line:

```
LD_LIBRARY_PATH=dist/lib ./yourcode omero.host=localhost --omero.user=foo --omero.
↪pass=bar
```

Note: This example explains how to run on Linux only. For doing the same on MacOS X, change all instances of LD_LIBRARY_PATH to DYLD_LIBRARY_PATH.

Further information

For the details behind writing, configuring, and executing a client, please see *Working with OMERO*.

See also:

Ice, *OMERO.grid*, *OMERO Application Programming Interface*, *Build System*, #1596 which added 64-bit support

3.2.7 JSON API

Overview

The OMERO JSON API described here provides create, read, update and delete access to an underlying OMERO server. It is implemented as a Django app named `api` in the OMERO.web framework.

Omoro-marshall and Projection-based APIs

The majority of the API URLs use `omero-marshall` to generate JSON dictionaries from OMERO **model** objects. All these URLs are under the `m` prefix:

```
<server>/api/v0/m/
```

The webclient currently uses a small number of URLs to perform customized queries for browsing Project, Dataset and Image hierarchies. These queries use **projections** and typically load a subset of fields for OMERO objects in order to improve performance for large data counts. These will be made available under the `p` prefix in future releases but are not yet supported.

```
<server>/api/v0/p/
```

Versioning

The JSON API uses major and minor version numbers to reflect breaking and non-breaking changes respectively. Non-breaking changes include simple addition of attributes to JSON data or addition of new URLs. The API version is not tied to the version of OMERO.server.

The major version is included in the URL such as `/v0/` whereas the full version number can be found in the header:

```
X-OMERO-ApiVersion : 0.2
```

JSON format

The JSON objects generated by `omero-marshall` are defined by the *OME-Model*. The OMERO model closely follows the OME schema but is not identical. In the cases where OMERO-specific fields are included, these will be prefixed by `omero:`. For example, `omero:details` specifies the owner, group and permissions of each object in OMERO. JSON objects also include an `@id` of the object in the OMERO database and a `@type` that specifies the OME Schema used to generate it such as `http://www.openmicroscopy.org/Schemas/OME/2016-06#Project`.

All the fields of the OMERO model object will be included in the JSON except those that are `null`, which will be omitted. Where supported, modifying the JSON object and saving this back to OMERO will update the object accordingly.

URLs in JSON

URLs are included in JSON objects using keys with the `url`: prefix. URLs are added for related objects to facilitate exploration of the API in a browser. You may find that a JSON formatting plugin for your browser improves both the presentation and navigation of JSON data.

Pagination

Requests that return a list of items will be paginated, showing a `limit` of the first 200 objects by default. Pagination can be specified using the `limit` and `offset` query parameters:

```
# List the first 100 Projects (offset=0 by default)
<server>/api/v0/m/projects/?limit=100

# List the next 100 Projects
<server>/api/v0/m/projects/?limit=100&offset=100
```

Pagination details will be returned in a meta JSON object, including the `totalCount` of objects for that query, the current `offset` and `limit` as well as the `maxLimit` that you can use.

```
"meta": {
  "totalCount": 13240,
  "maxLimit": 500,
  "limit": 200,
  "offset": 0
},
```

Sysadmins can configure the default `limit` and `maxLimit` settings for their server, for example:

```
$ omero config set omero.web.api.limit 100
$ omero config set omero.web.api.max_limit 300
```

The `maxLimit` setting prevents API consumers from requesting very large amounts of data by limiting the number of top-level objects that are loaded.

Loading of linked objects

In most cases the API loads only the requested objects along with their `omero:details`. For example, `/api/v0/m/projects/` loads Projects but does not also load their child Datasets. However, it is sometimes useful to load a number of closely related objects. For example, loading Images also loads their Pixels data (but not Channels) and loading Wells also loads WellSamples (fields) and Images (but not Pixels). The number of objects loaded when listing Images or Wells is kept to a minimum to avoid requesting too much data. This restriction is relaxed when a single Image or Well is loaded. For example, loading a single Image will also load Channels.

Normalizing Experimenters and Groups

When returning a list of JSON objects that each contain `omero:details` with `owner` and `group` data, these will typically be nested many times within the list. In order to avoid this duplication, we can remove objects from within each `omero:details` and place them under top-level `experimenters` and `experimenterGroups` lists. You can specify this with the `?normalize=true` query parameter. N.B.: Currently this normalizing will only apply to the top-level objects being listed, such as Projects, Datasets and Images. Where child objects are also loaded (for example Pixels within an Image), the `omero:details` of these objects will not be affected by the `?normalize=true` parameter.

Child counts

For container objects such as Projects, Datasets and Screens it is often useful to know the number of children within them. This can be specified with `?childCount=true` parameter. This will add an `omero:childCount` value to the JSON data.

Filtering by Owner and Group

Most data in OMERO has an `Owner` and is assigned to a permissions `Group`. By default, queries will return data from all owners across all groups that are accessible to the current user. Use the query strings to filter by owner and/or group:

```
/api/v0/m/projects/?owner=3&group=5
```

When you are retrieving data using an object ID you will not need to filter by `group` since all the data will be in the same group. For example, Datasets in a specified Project will all be in the same group as the Project.

Error handling

Errors will result in responses with an appropriate status and may include JSON content with a `message` to provide more information:

- **404 Not Found:** Caused by an invalid URL or when a specified object cannot be found in OMERO.
- **400 Bad Request:** May be caused by invalid query parameters or submitting invalid JSON content. For example, `?limit=foo` will give a response of:

```
{"message": "invalid literal for int() with base 10: 'foo'"}
```

- **405 Method Not Allowed:** Returned if you try to use the wrong http method for a url, such as POST to `/api/v0/m/projects/`. It can also be caused by trying to create or update an unsupported object, such as an Image.
- **500 Internal Server Error:** Generated from any unhandled exceptions. See the `message` returned and check whether a `stacktrace` is also included.

Getting started

You may find this [example python script](#) useful. It uses the python `requests` library to connect to the JSON api, login, query data, create and delete Projects. These steps are covered in more detail below.

For an example how to use the API with Java, see `JSONClient.java`.

See the following link for a JSON client example <https://github.com/ome/openmicroscopy/blob/develop/examples/Training/javascript/index.html>.

List supported versions

You need to find which versions of the API are supported by your server, as described above. These are provided by the base URL:

```
GET      /api/
```

Response

```
{
  "data": [
    {
      "version": "0",
      "url:base": "http://<server>/api/v0/"
    }
  ]
}
```

List starting URLs

The base URL for the chosen version will list a number of URLs for logging on and getting started.

```
GET      /api/v0/
```

Response

```
{
  "url:login": "http://<server>/api/v0/login/",
  "url:save": "http://<server>/api/v0/m/save/",
  "url:projects": "http://<server>/api/v0/m/projects/",
  "url:plates": "http://<server>/api/v0/m/plates/",
  "url:datasets": "http://<server>/api/v0/m/datasets/",
  "url:token": "http://<server>/api/v0/token/",
  "url:schema": "http://www.openmicroscopy.org/Schemas/OME/2016-06",
  "url:screens": "http://<server>/api/v0/m/screens/",
  "url:servers": "http://<server>/api/v0/servers/",
  "url:images": "http://<server>/api/v0/m/images/"
}
```


List available OMERO servers

Your API may allow you to connect to several different OMERO servers.

GET /api/v0/servers/

Response

```
{
  "data": [
    {
      "host": "<server>",
      "server": "omero",
      "id": 1,
      "port": 4064
    }
  ]
}
```

Get CSRF token

In order to prevent CSRF attacks, CSRF tokens are required for any POST, PUT and DELETE requests. You will need to obtain a CSRF token for your session and use it for all subsequent requests in that session. You can obtain this from the `csrftoken` cookie of any request or from the following URL:

GET /api/v0/token/

Response

```
{
  "data": "eNoVq528bOqlhQqbCzKuviODTRX3PU02"
}
```

Login

You can login to create an OMERO session. You must also include the CSRF token, either in the POST parameters as `csrftoken` or in the session header as `X-CSRFToken`.

The EventContext for this session will be returned to you.

POST /api/v0/login/

Parameters

Name	Type	Description
server	Number	ID of the server
username	String	User's <code>username</code>
password	String	User's <code>password</code>
csrftoken	String	CSRF token (can be provided <code>in</code> header)

Response

```
{
  "eventContext": {
    "userName": "ben",
    "eventId": -1,
    "sessionId": "0b30ee4a-c0b2-4b0f-9c61-f48b31bcad8c",
    "eventType": "User",
    "userId": 3,
    "sessionId": 171319,
    "groupName": "Nevis Lab",
    "isAdmin": false,
    "memberOfGroups": [5, 1, 4],
    "leaderOfGroups": [],
    "groupId": 5
  },
  "success": true
}
```

Projects, Datasets and Images

OMERO organizes Images in two types of many-to-many hierarchy: `screen/plate/[run]/well/image` for HCS data and `project/dataset/image` for other data. Plates, Datasets and Images can also be Orphaned if not contained within any parent container.

Parameters

These query parameters are used by many queries below:

Name	Type	Description
offset	Number	Pagination offset. The default is 0
limit	Number	The size of each page. The default is 200
normalize	Boolean	Place Experimenters and Groups into top-level lists instead of nesting within objects
childCount	Boolean	Use ?childCount=true to include an <code>omero:childCount</code> attribute for container objects
owner	Number	Filter by Experimenter ID
group	Number	Filter by Group ID

List Projects

Parameters

Name	Type	Description
dataset	Number	Filter Projects by child Dataset ID

These query parameters are also supported (see above):

```
offset, limit, owner, group, childCount, normalize
```

```
GET /api/v0/m/projects/
```

Response

```
{
  "data": [
    {
      "Name": "New data",
      "Description": "Example Project",
      "url:project": "https://server.openmicroscopy.org/api/v0/m/projects/11601/",
      "url:datasets": "https://server.openmicroscopy.org/api/v0/m/projects/11601/
↳ datasets/",
      "@id": 11601,
      "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Project",
      "omero:details": {
        "owner": {
          "UserName": "ben",
          "FirstName": "Ben",
          "MiddleName": "",
          "omero:details": {
            "@type": "TBD#Details",
            "permissions": {
              "isUserWrite": false,
              "isWorldWrite": false,
              "canDelete": false,
              "isWorldRead": false,
              "perm": "-----",
              "canEdit": false,
              "canAnnotate": false,
              "isGroupAnnotate": false,
              "isGroupWrite": false,
              "canLink": false,
              "isUserRead": false,
              "@type": "TBD#Permissions",
              "isGroupRead": false
            }
          }
        },
        "Email": "",
        "LastName": "Nevis",
        "@id": 0,
        "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Experimenter"
      },
      "group": {
        "omero:details": {
          "@type": "TBD#Details",
          "permissions": {
            "isUserWrite": true,
            "isWorldWrite": false,
            "canDelete": false,
            "isWorldRead": false,
            "perm": "rwra--",
```

(continues on next page)

(continued from previous page)

```

        "canEdit": false,
        "canAnnotate": false,
        "isGroupAnnotate": true,
        "isGroupWrite": false,
        "canLink": false,
        "isUserRead": true,
        "@type": "TBD#Permissions",
        "isGroupRead": true
    }
},
"@id": 5,
"@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#ExperimenterGroup",
"Name": "read-ann"
},
"@type": "TBD#Details",
"permissions": {
    "isUserWrite": true,
    "isWorldWrite": false,
    "canDelete": false,
    "isWorldRead": false,
    "perm": "rwra--",
    "canEdit": false,
    "canAnnotate": true,
    "isGroupAnnotate": true,
    "isGroupWrite": false,
    "canLink": false,
    "isUserRead": true,
    "@type": "TBD#Permissions",
    "isGroupRead": true
}
}
}
]
}

```

Get a single Project

```
GET /api/v0/m/projects/{project_id}/
```

Response

```

{
  "data": {
    "@id": 3872,
    "Name": "RNAi experiments",
    "Description": "Knockout assays",
    "url:datasets": "https://server.openmicroscopy.org/api/v0/m/projects/3872/datasets/",
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Project",
    "omero:details": {
      # omitted for brevity
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

    }
  }
}

```

List Datasets

Parameters

Name	Type	Description
project	Number	Filter Datasets by parent Project ID
image	Number	Filter Datasets by child Image ID
orphaned	Boolean	Find Datasets that are not in any Project

These query parameters are also supported (see above):

```
offset, limit, owner, group, childCount, normalize
```

```
GET    /api/v0/m/datasets/
```

Response

```

{
  "data": [
    {
      "Name": "Test data",
      "Description": "This is the Dataset description",
      "url:dataset": "https://server.openmicroscopy.org/api/v0/m/dataset/112/",
      "url:images": "https://server.openmicroscopy.org/api/v0/m/datasets/112/images/",
      "url:projects": "https://server.openmicroscopy.org/api/v0/m/datasets/112/projects/
→",
      "@id": 112,
      "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Project",
      "omero:details": {
        # omitted for brevity
      }
    }
  ]
}

```

Datasets in a Project

Datasets can be filtered by parent Project using the `?project=id` query string but you can also show Datasets in a Project using this URL:

```
GET    /api/v0/m/projects/{project_id}/datasets/
```

Get a single Dataset

```
GET /api/v0/m/datasets/{dataset_id}/
```

Response

```
{
  "data": {
    "@id": 9702,
    "Name": "My data",
    "Description": "An example set",
    "url:images": "https://server.openmicroscopy.org/api/v0/m/datasets/9702/images/",
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Dataset",
    "omero:details": {
      # omitted for brevity
    }
  }
}
```

List Images

When Images are listed, their `Pixels` object is also loaded, which includes dimensions and pixel sizes of the Image. When a single Image is retrieved, the `Channels` data is additionally loaded.

Parameters

Name	Type	Description
dataset	Number	Filter Images by parent Dataset ID
orphaned	Boolean	Find Images that are not in any Dataset or Well

These query parameters are also supported (see above):

```
offset, limit, owner, group, normalize
```

```
GET /api/v0/m/images/
```

Response

```
{
  "data": [
    {
      "@id": 16783,
      "Name": "CFP_AurB_R3D.dv",
      "AcquisitionDate": 1235730332000,
      "omero:details": {
        # omitted for brevity
      },
      "url:image": "https://server.openmicroscopy.org/api/v0/m/images/16783/",
      "Pixels": {
        "@id": 12801,
```

(continues on next page)

(continued from previous page)

```

    "SizeX": 512,
    "SizeY": 512,
    "SizeZ": 29,
    "SizeC": 2,
    "SizeT": 1,
    "PhysicalSizeX": {
      "Symbol": "µm",
      "Value": 0.12698,
      "@type": "TBD#LengthI",
      "Unit": "MICROMETER"
    },
    "PhysicalSizeY": {
      "Symbol": "µm",
      "Value": 0.12698,
      "@type": "TBD#LengthI",
      "Unit": "MICROMETER"
    },
    "PhysicalSizeZ": {
      "Symbol": "µm",
      "Value": 0.2,
      "@type": "TBD#LengthI",
      "Unit": "MICROMETER"
    },
    "Type": {
      "omero:details": {
        # omitted for brevity
      },
      "@id": 6,
      "@type": "TBD#PixelsType",
      "value": "uint16"
    },
    "omero:sha1": "eae01c54191fd9cf4b09e3651e1899d677375b7d",
    "omero:details": {
      # omitted for brevity
    },
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Pixels",
    "SignificantBits": 16
  },
  "omero:series": 0,
  "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Image"
}
]
}

```

Images in a Dataset

Images can be filtered by parent Dataset using the `?dataset=id` query string but you can also show Images in a Dataset using this URL:

```
GET /api/v0/m/datasets/{dataset_id}/images/
```

Get a single Image

```
GET /api/v0/m/images/{image_id}/
```

Response

The response for a single Image is the same as for listing Images above with the addition of Channels data.

```
{
  "data": [
    {
      "@id": 16783,
      "Name": "CFP_AurB_R3D.dv",
      "AcquisitionDate": 1235730332000,
      "omero:details": {
        # omitted for brevity
      },
      "Pixels": {
        "@id": 12801,
        "Channels": [
          {
            "omero:photometricInterpretation": {
              "omero:details": {},
              "@id": 5,
              "@type": "TBD#PhotometricInterpretation",
              "value": "Monochrome"
            },
            "Name": "CFP_JP4",
            "Color": 65535,
            "omero:details": {},
            "ExcitationWavelength": {
              "Symbol": "nm",
              "Value": 436,
              "@type": "TBD#LengthI",
              "Unit": "NANOMETER"
            },
            "SamplesPerPixel": 1,
            "NDFilter": 1,
            "EmissionWavelength": {
              "Symbol": "nm",
              "Value": 470,
              "@type": "TBD#LengthI",
              "Unit": "NANOMETER"
            },
            "omero:LogicalChannelId": 12301,
            "@id": 14451,
            "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Channel"
          },
          {
            "omero:photometricInterpretation": {
              "omero:details": {},
              "@id": 5,
              "@type": "TBD#PhotometricInterpretation",
```

(continues on next page)

(continued from previous page)

```

        "value": "Monochrome"
    },
    "Name": "RD_TR-PE",
    "Color": -16776961,
    "omero:details": {},
    "ExcitationWavelength": {
        "Symbol": "nm",
        "Value": 555,
        "@type": "TBD#LengthI",
        "Unit": "NANOMETER"
    },
    "SamplesPerPixel": 1,
    "NDFilter": 0,
    "EmissionWavelength": {
        "Symbol": "nm",
        "Value": 617,
        "@type": "TBD#LengthI",
        "Unit": "NANOMETER"
    },
    "omero:LogicalChannelId": 12303,
    "@id": 14453,
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Channel"
}
],
"SizeX": 512,
"SizeY": 512,
"SizeZ": 29,
"SizeC": 2,
"SizeT": 1,
"PhysicalSizeX": {
    "Symbol": "µm",
    "Value": 0.12698,
    "@type": "TBD#LengthI",
    "Unit": "MICROMETER"
},
"PhysicalSizeY": {
    "Symbol": "µm",
    "Value": 0.12698,
    "@type": "TBD#LengthI",
    "Unit": "MICROMETER"
},
"PhysicalSizeZ": {
    "Symbol": "µm",
    "Value": 0.2,
    "@type": "TBD#LengthI",
    "Unit": "MICROMETER"
},
"Type": {
    "omero:details": {
        # omitted for brevity
    },
    "@id": 6,

```

(continues on next page)

(continued from previous page)

```

        "@type": "TBD#PixelsType",
        "value": "uint16"
    },
    "omero:sha1": "eae01c54191fd9cf4b09e3651e1899d677375b7d",
    "omero:details": {
        # omitted for brevity
    },
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Pixels",
    "SignificantBits": 16
},
"omero:series": 0,
"@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Image"
}
]
}

```

Screens, Plates and Wells

For more information on the Screen, Plate, Well data model, please see the [documentation page](#).

List Screens

Parameters

Name	Type	Description
plate	Number	Filter Datasets by child Plate ID

These query parameters are also supported (see above):

offset, limit, owner, group, childCount, normalize

GET /api/v0/m/screens/

Response

```

{
  "data": [
    {
      "@id": 582,
      "Name": "Test data",
      "Description": "This is the Screen description",
      "url:screen": "https://server.openmicroscopy.org/api/v0/m/screen/582/",
      "url:plates": "https://server.openmicroscopy.org/api/v0/m/screen/582/plates/",
      "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Screen",
      "omero:details": {
        # omitted for brevity
      }
    }
  ]
}

```

Get a single Screen

```
GET /api/v0/m/screens/{screen_id}/
```

Response

```
{
  "data": {
    "@id": 582,
    "Name": "Test data",
    "Description": "This is the Screen description",
    "url:plates": "https://server.openmicroscopy.org/api/v0/m/screen/582/plates/",
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Screen",
    "omero:details": {
      # omitted for brevity
    }
  }
}
```

List Plates

Parameters

Name	Type	Description
screen	Number	Filter Plates by parent Screen ID
well	Number	Filter Plates by child Well ID
orphaned	Boolean	Find Plates that are not in any Screen

These query parameters are also supported (see above):

```
offset, limit, owner, group, childCount, normalize
```

```
GET /api/v0/m/plates/
```

Response

```
{
  "data": [
    {
      "@id": 5067,
      "Name": "Plate name",
      "Rows": 8,
      "Columns": 12,
      "RowNamingConvention": "letter",
      "ColumnNamingConvention": "number",
      "ExternalIdentifier": "003857",
      "url:plate": "https://server.openmicroscopy.org/api/v0/m/plates/5067/",
      "url:plateacquisitions": "https://server.openmicroscopy.org/api/v0/m/plates/5067/
↪plateacquisitions/",
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    "url:wells": "https://server.openmicroscopy.org/api/v0/m/plates/5067/wells/",
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Plate",
    "omero:details": {
      # omitted for brevity
    },
  },
]
}

```

Plates in a Screen

Plates can be filtered by parent Screen using the `?screen=id` query string but you can also show Plates in a Screen using this URL:

```
GET /api/v0/m/screens/{screen_id}/plates/
```

Get a single Plate

```
GET /api/v0/m/plates/{plate_id}/
```

Response

The response for a single Plate includes information on the WellSamples (fields) for each Well such as the min/max WellSampleIndex for the Plate.

```

{
  "data": {
    "@id": 5067,
    "Name": "Plate name",
    "Rows": 8,
    "Columns": 12,
    "RowNamingConvention": "letter",
    "ColumnNamingConvention": "number",
    "ExternalIdentifier": "003857",
    "url:plate": "https://server.openmicroscopy.org/api/v0/m/plates/5067/",
    "url:plateacquisitions": "https://server.openmicroscopy.org/api/v0/m/plates/5067/
    ↪plateacquisitions/",
    "url:wells": "https://server.openmicroscopy.org/api/v0/m/plates/5067/wells/",
    "url:wellsampleindex_wells": [
      "https://server.openmicroscopy.org/api/v0/m/plates/5068/wellsampleindex/0/wells/",
      "https://server.openmicroscopy.org/api/v0/m/plates/5068/wellsampleindex/1/wells/",
      "https://server.openmicroscopy.org/api/v0/m/plates/5068/wellsampleindex/2/wells/",
      "https://server.openmicroscopy.org/api/v0/m/plates/5068/wellsampleindex/3/wells/"
    ],
    "omero:wellsampleIndex": [
      0,
      3
    ],
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Plate",
    "omero:details": {
      # omitted for brevity
    }
  }
}

```

(continues on next page)

(continued from previous page)

```
}
}
```

List Plate Acquisitions

A Plate Acquisition (run) is a collection of WellSamples, grouped by an acquisition time. A Plate may contain zero, one or more Plate Acquisitions.

```
GET /api/v0/m/plates/{plate_id}/plateacquisitions/
```

Response

```
{
  "data": [
    {
      "@id": 4217,
      "url:wellsampleindex_wells": [
        "https://server.openmicroscopy.org/api/v0/m/plateacquisitions/4217/
↪wellsampleindex/0/wells/"
        "https://server.openmicroscopy.org/api/v0/m/plateacquisitions/4217/
↪wellsampleindex/1/wells/"
        "https://server.openmicroscopy.org/api/v0/m/plateacquisitions/4217/
↪wellsampleindex/2/wells/"
      ],
      "omero:details": {
        # omitted for brevity
      },
      "MaximumFieldCount": 3,
      "url:plateacquisition": "https://server.openmicroscopy.org/api/v0/m/
↪plateacquisitions/4217/",
      "omero:wellsampleIndex": [
        0,
        2
      ],
      "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#PlateAcquisition"
    }
  ]
}
```

List Wells in a Plate

Each Well in a Plate may contain zero, one or many WellSamples (fields). By default, when listing Wells in a Plate, *all* of the WellSamples and Images will be loaded for each Well. Wells are ordered by Column and Row.

Parameters

The following query parameters can be used (as described above)

```
offset, limit, owner, normalize
```

```
GET /api/v0/m/plates/{plate_id}/wells/
```

Note: If there are a large number of WellSamples per Well, this has the potential to load a large amount of data. This can be reduced by using a smaller `limit` on the number of Wells loaded or only loading a single WellSample per Well, as described below.

Response

```
{
  "data": [
    {
      "@id": 139,
      "Column": 0,
      "Row": 0,
      "omero:details": {
        # omitted for brevity
      },
      "url:well": "https://server.openmicroscopy.org/api/v0/m/wells/139/",
      "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Well",
      "WellSamples": [
        {
          "PositionX": {
            "Symbol": "reference frame",
            "Value": 21864.47,
            "@type": "TBD#LengthI",
            "Unit": "REFERENCEFRAME"
          },
          "PositionY": {
            "Symbol": "reference frame",
            "Value": 36711.98,
            "@type": "TBD#LengthI",
            "Unit": "REFERENCEFRAME"
          },
          "omero:details": {
            # omitted for brevity
          },
          "Image": {
            "Name": "plate1.HTD [Well E02 Field #1]",
            "AcquisitionDate": 1252939626000,
            "omero:details": {
              # omitted for brevity
            },
            "url:image": "https://server.openmicroscopy.org/api/v0/m/images/2942/",
            "omero:series": 120,
            "@id": 2942,
            "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Image",
            "Description": "Scan Time: Mon Sep 14 11:36:58 2009"
          },
          "PlateAcquisition": {
            "omero:details": {
              # omitted for brevity
            }
          }
        }
      ]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    },
    "MaximumFieldCount": 4,
    "StartTime": 1252938959000,
    "EndTime": 1252939813000,
    "@id": 102,
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#PlateAcquisition"
  },
  "@id": 203,
  "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#WellSample"
}
]
}
]
}

```

It is also possible to list all Wells without filtering by Plate, using the top-level URL `/api/v0/m/wells/` optionally filtering by the `plate` query parameter.

List Wells by WellSample Index

To list Wells in a Plate, loading only a *single* WellSample and Image per Well, you can filter by WellSample Index. This list of Wells will not include empty Wells (Wells that have no WellSamples and Images).

```
GET /api/v0/m/plates/{plate_id}/wellsampleindex/{index}/wells/
```

It is also possible to use the Plate Acquisition ID instead of Plate ID, when the WellSample (field) at the specified index was acquired as part of that Plate Acquisition:

```
GET /api/v0/m/plateacquisitions/{plateacquisition_id}/wellsampleindex/{index}/wells/
```

Get a single Well

When a single Well is loaded, this will include all the WellSamples and Images with Pixels loaded.

```
GET /api/v0/m/wells/{well_id}/
```

ROIs and Shapes

Support for listing ROIs was added in API version 0.1. ROIs are linked to Images and contain one or more Shapes. Types of shape are Ellipse, Label, Line, Mask, Point, Polygon, Polyline and Rectangle.

List ROIs

When ROIs are listed, their child Shapes will also be loaded.

Parameters

Name	Type	Description
image	Number	Filter ROIs by Image ID

These query parameters are also supported (see above):

offset, limit, owner, group, normalize

GET /api/v0/m/rois/

Response

```
{
  "data": [
    {
      "@id": 454,
      "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#ROI",
      "shapes": [
        {
          "FontStyle": "Normal",
          "Locked": false,
          "Width": 98,
          "omero:details": {
            # omitted for brevity
          },
          "Height": 135,
          "FontFamily": "sans-serif",
          "StrokeWidth": {},
          "FontSize": {
            "Symbol": "pt",
            "Value": 12,
            "@type": "TBD#LengthI",
            "Unit": "POINT"
          },
          "FillColor": 1073741824,
          "Y": 192,
          "X": 189,
          "StrokeColor": -993737532,
          "TheT": 23,
          "@id": 713,
          "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Rectangle",
          "TheZ": 1
        }
      ],
      "omero:details": {
        # omitted for brevity
      },
    }
  ],
}
```

(continues on next page)

(continued from previous page)

```
]
}
```

ROIs on an Image

ROIs can be filtered by Image using the `?image=id` query string but you can also show ROIs on an Image using this URL:

```
GET      /api/v0/m/images/{image_id}/rois/
```

Experimenters and Groups

Support for listing Experimenters and Groups was added in API version 0.2. Experimenters are users of OMERO and can belong to one or more Groups. Groups are defined as ExperimenterGroups in the OME model.

Listing Experimenters

OMERO will only allow you to access details of Experimenters who are members of a **non-private** group that you are also a member of.

Parameters

Name	Type	Description
experimentergroup	Number	Filter Experimenters by Group

These query parameters are also supported (see above):

```
offset, limit
```

```
GET      /api/v0/m/experimenters/
```

Response

```
{
  "data": [
    {
      "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Experimenter",
      "@id": 10,
      "omero:details": {
        "@type": "TBD#Details",
        "permissions": {
          "@type": "TBD#Permissions",
          "perm": "-----",
          "canAnnotate": true,
          "canDelete": false,
          "canEdit": false,
          "canLink": true,
          "isWorldWrite": false,
          "isWorldRead": false,
          "isGroupWrite": false,

```

(continues on next page)

(continued from previous page)

```

        "isGroupRead": false,
        "isGroupAnnotate": false,
        "isUserWrite": false,
        "isUserRead": false
      },
      {
        "FirstName": "Ben",
        "LastName": "Nevis",
        "UserName": "ben",
        "url:experimenter": "https://server.openmicroscopy.org/web/api/v0/m/experimenters/
↪10/",
        "url:experimentergroups": "https://server.openmicroscopy.org/web/api/v0/m/
↪experimenters/10/experimentergroups/"
      },
    ],
  ]
}

```

Get a single Experimenter

Load an Experimenter with:

```
GET /api/v0/m/experimenters/{experimenter_id}/
```

Experimenters in a Group

Experimenters can be filtered by Group using the `?experimentergroup=id` query string but you can also show Members of a Group using this URL:

```
GET /api/v0/m/experimentergroups/{group_id}/experimenters/
```

Listing Groups

Parameters

Name	Type	Description
experimenter	Number	Filter Groups by Experimenter

These query parameters are also supported (see above):

```
offset, limit
```

```
GET /api/v0/m/experimentergroups/
```

Response

```
{
  "data": [
```

(continues on next page)

(continued from previous page)

```

{
  "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#ExperimenterGroup",
  "@id": 10,
  "omero:details": {
    "@type": "TBD#Details",
    "permissions": {
      "@type": "TBD#Permissions",
      "perm": "-----",
      "canAnnotate": true,
      "canDelete": false,
      "canEdit": false,
      "canLink": true,
      "isWorldWrite": false,
      "isWorldRead": false,
      "isGroupWrite": false,
      "isGroupRead": false,
      "isGroupAnnotate": false,
      "isUserWrite": false,
      "isUserRead": false
    }
  },
  "Name": "Swedlow Lab",
  "url:experimentergroup": "https://server.openmicroscopy.org/web/api/v0/m/
↪ experimentergroups/10/",
  "url:experimenters": "https://server.openmicroscopy.org/web/api/v0/m/
↪ experimentergroups/10/experimenters/"
},
]
}

```

Get a single Group

Load a Group with:

```
GET /api/v0/m/experimentergroups/{group_id}/
```

Groups for an Experimenter

Groups can be filtered by Experimenter using the `?experimenter=id` query string but you can also show ExperimenterGroups that an Experimenter belongs to using this URL:

```
GET    /api/v0/m/experimenters/{experimenter_id}/experimentergroups
```

Creating and saving objects

The JSON API currently supports creating and saving of a limited number of object types, namely Projects, Datasets and Screens. It is not yet possible to save objects with unloaded objects, such as an Image without Pixels or Channels loaded. We will be working to resolve these issues in future releases.

Creating and saving of JSON objects are handled by a single `save` URL and objects are identified by their `@type` and `@id` attributes.

Object types

The object `@type` must be based on the currently supported Schema URL which can be retrieved with:

```
GET    /api/v0/
```

Response

```
{
  "url:schema": "http://www.openmicroscopy.org/Schemas/OME/2016-06",
  # other urls not shown
}
```

This can then be used to create a `@type` by appending `#` and the object name, such as:

```
http://www.openmicroscopy.org/Schemas/OME/2016-06#Project
```

Creating objects

To create an object, POST the JSON for that object, including the ID of the OMERO group that the object should be saved in. Currently only creation of Projects, Datasets and Screens is supported.

```
POST   /api/v0/m/save/?group={group_id}
```

Content

```
{
  "Name": "My new Project",
  "Description": "Created via the JSON API",
  "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Project"
}
```

Response

```
{
  "data": {
    "@id": 567,
    "Name": "My new Project",
    "Description": "Created via the JSON API",
    "url:datasets": "https://server.openmicroscopy.org/api/v0/m/projects/3872/datasets/",
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Project",
    "omero:details": {
      # omitted for brevity
    }
  }
}
```

Updating objects

The API supports PUT to replace existing objects with the submitted data. As mentioned above, the only objects that you can currently update are Projects, Datasets and Screens. The submitted JSON data can be constructed from scratch, but it will generally be more convenient and safer to GET the object, update it and save the edited JSON.

For example, to edit the Name of the Project in the previous example:

```
PUT /api/v0/m/save/
```

Content

```
{
  "@id": 567,
  "Name": "Edited Project Name",
  "Description": "Created via the JSON API",
  "url:datasets": "https://server.openmicroscopy.org/api/v0/m/projects/3872/datasets/",
  "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Project",
  "omero:details": {
    # omitted for brevity
  }
}
```

Response

```
{
  "data": {
    "@id": 567,
    "Name": "Edited Project Name",
    "Description": "Created via the JSON API",
    "url:datasets": "https://server.openmicroscopy.org/api/v0/m/projects/3872/datasets/",
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Project",
    "omero:details": {
      # omitted for brevity
    }
  }
}
```

Deleting objects

To delete a Project, Dataset or Screen, simply DELETE using the URL to that object. The deleted object will be returned. For example, to delete a Project:

```
DELETE /api/v0/m/projects/{project_id}/
```

Response

```
{
  "data": {
    "@id": 567,
    "Name": "Edited Project Name",
    "Description": "Created via the JSON API",
    "url:datasets": "https://server.openmicroscopy.org/api/v0/m/projects/3872/datasets/",
    "@type": "http://www.openmicroscopy.org/Schemas/OME/2016-06#Project",
    "omero:details": {
      # omitted for brevity
    }
  }
}
```

3.3 Analysis

3.3.1 Local analysis

If you are interested in running your analysis locally and storing the results to the server, then your first step is to become familiar with the developer documentation.

- The *Working with OMERO* guide provides numerous examples in each language with explanations and tries to be a starting point for anyone who wants to write code which talks to the OMERO server.
- Most of the *OMERO Application Programming Interface* is covered by the *Javadocs*.
- Each of the languages has extra information on its own page:
 - *OMERO C++ language bindings*
 - *OMERO Java language bindings*
 - *OMERO MATLAB language bindings*
 - *OMERO Python language bindings*

Once you have your local analysis working, you can push it onto the server for background processing using the *OMERO scripting service*.

3.3.2 Storing external data in OMERO

There are several options for storing external or schema-less data in OMERO, including *StructuredAnnotations* for small quantities of data, or extending the OME model, but this risks interoperability issues. (See *Extending Omero*).

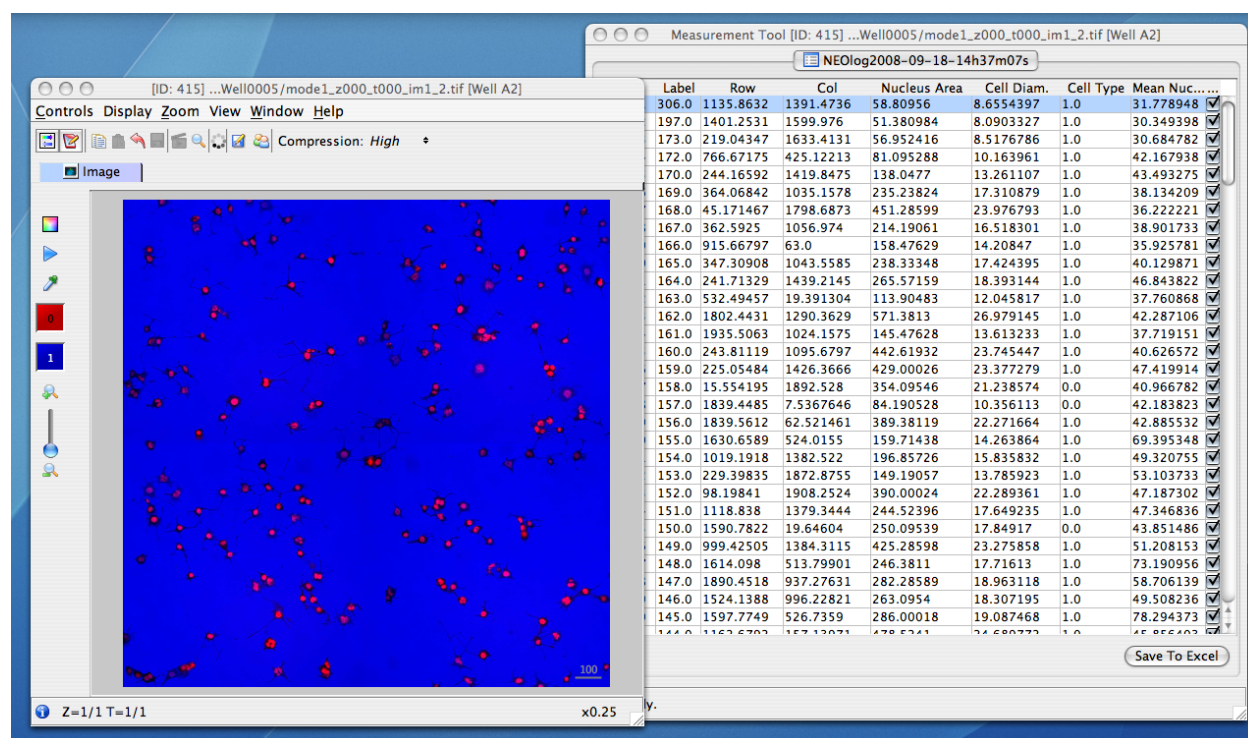
For larger volumes of data, or data which needs to be queried, *OMERO.tables* provides a unified and effective solution for the storage of tabular data from various sources, such as automated analysis results or script-based processing, and makes them available within OMERO.

Third-party analysis and OMERO.tables

Support has been added for some third-party analysis data, which gets converted in OMERO into a common format. These formats include:

- MIAS data, measurements, and overlays
- InCell data and measurements
- Flex data with Acapella results ([screencast](#)). In the Flex case, additional configuration may be necessary for accessing both the raw data and the analysis results. Watch [the configuration screencast](#) for more information.

The analysis results which are parsed out of the formats listed above are converted to HDF by the OMERO.tables API. This facility can then be used by clients to visualize the parsed measurements, and in the case of regions of interest, see their location overlaid on the associated image:



Other high-content screening (HCS) data

In addition to the Flex, Mias, and InCell 100 file formats, BD Pathway, Olympus ScanR, and native OME-XML/TIFF files can all be imported as HCS data, though without support for any external analysis data which may be attached. If you are interested in having other analysis formats supported, contact either the [open source community](#) or [Glencoe Software, Inc.](#) depending on your needs.

3.3.3 OMERO.tables

The OMERO.tables API unifies the storage of columnar data from various sources, such as automated analysis results or script-based processing, and makes them available within OMERO.

Large and small volumes of tabular data can be stored via named columns, and retrieved in bulk or via paging. A limited query language provides basic filtering and selecting.

Since 5.6, the client library `omero-py` is available on [PyPI](#) and [Conda](#). We recommend to install the library in a Python virtual environment. In the same environment, you should now install [PyTables](#) by running:

```
$ pip install tables
```

Note that PyTables has dropped support for Python 3.8 in the 3.9.x line, see [tag v3.9.1](#).

The interface

The [slice definition file](#) for the OMERO.tables API primarily defines two service interfaces and a type hierarchy.

class `omero.grid.Table`

The central service for dealing with tabular data, described [below](#).

class `omero.grid.Tables`

An internal service used for managing table services, and can be ignored for almost all purposes.

class `omero.grid.Column`

The base class for column types which permit returning arrays of columnar values ([Ice](#) doesn't provide an `Any` type, so it is necessary to group values of the same type). All columns in a table must have the same number of rows.

Note: Attribute names (including column names) beginning with `__` (double underscore) are reserved for internal use. This restriction was introduced in OMERO 5.1, Tables created by older versions should continue to work.

Single value columns

These columns store a single value in each row.

```
class omero.grid.FileColumn(name, description[, values])
class omero.grid.ImageColumn(name, description[, values])
class omero.grid.RoiColumn(name, description[, values])
class omero.grid.WellColumn(name, description[, values])
```


class omero.grid.**PlateColumn**(*name*, *description*[, *values*])

Id-based (*long*) columns which reference `omero.model.File`, `Image`, `Roi`, `Well` and `Plate` instances respectively.

class omero.grid.**BoolColumn**(*name*, *description*[, *values*])

A value column with *bool* (non-null) values.

class omero.grid.**LongColumn**(*name*, *description*[, *values*])

A value column with *long* (non-null, 64-bit) values.

class omero.grid.**DoubleColumn**(*name*, *description*[, *values*])

A value column with *double* (non-null, 64-bit) values.

Parameters

- **name** (*string*) – The name of the column, each column in a table must have a unique name.
- **description** (*string*) – The column description, may be empty.
- **values** ([*l*]) – A list of values (one value per row) used to initialize a column (optional).

values

A class member holding the list of values stored in the column.

class omero.grid.**StringColumn**(*name*, *description*, *size*[, *values*])

A value column which holds strings

Parameters

- **name** (*string*) – The column name.
- **description** (*string*) – The column description.
- **size** (*long*) – The maximum string length that can be stored in this column, ≥ 1
- **values** (*string*[*l*]) – A list of strings (optional).

Array value columns

These columns store an array in each row.

class omero.grid.**FloatArrayColumn**(*name*, *description*, *size*[, *values*])

A value column with fixed-width arrays of *float* (32 bit) values.

class omero.grid.**DoubleArrayColumn**(*name*, *description*, *size*[, *values*])

A value column with fixed-width arrays of *double* (64 bit) values.

class omero.grid.**LongArrayColumn**(*name*, *description*, *size*[, *values*])

A value column with fixed-width arrays of *long* (64 bit) values.

Parameters

- **name** (*string*) – The column name.
- **description** (*string*) – The column description.
- **size** (*long*) – The width of the array, ≥ 1
- **values** ([*l*][*l*]) – A list of arrays, each of length *size* (optional).

Warning: The OMERO.tables service currently does limited validation of string and array lengths. When adding or modifying data it is essential that the `size` parameter of a column matches that of the underlying table.

Warning: Array value columns should be considered experimental for now.

Main methods

class omero.grid.Data

Holds the data retrieved from a table, also used to update a table.

lastModification

The timestamp of the last update to the table.

rowNumbers

The row indices of the values retrieved from the table.

columns

A list of columns

class omero.grid.Table

The main interface to the Tables service.

getHeaders()

Returns

An empty list of columns describing the table. Fill in the values of these columns to add a new row to the table.

getNumberOfRows()

Returns

The number of rows in the table.

readCoordinates(*rowNumbers*)

Read a set of entire rows in the table.

Parameters

rowNumbers (*long*[]) – A list of row indices to be retrieved from the table.

Returns

The requested rows as a [Data](#) object.

read(*colNumbers*, *start*, *stop*)

Read a subset of columns and consecutive rows from a table.

Parameters

- **colNumber** (*long*[]) – A list of column indices to be retrieved from the table (may be non-consecutive).
- **start** (*long*) – The index of the first row to retrieve.
- **stop** (*long*) – The index of the *last+1* row to retrieve (uses similar semantics to `range()`).

Returns

The requested columns and rows as a [Data](#) object.

Note: `start=0, stop=0` currently returns the first row instead of empty as would be expected using the normal Python range semantics. This may change in future.

slice(*colNumbers*, *rowNumbers*)

Read a subset of columns and rows (may be non-consecutive) from a table.

Parameters

- **colNumbers** (*long*[]) – A list of column indices to be retrieved. The results will be returned in the same order as these indices.
- **rowNumbers** (*long*[]) – A list of row indices to be retrieved. The results will be returned in the same order as these indices.

Returns

The requested columns and rows as a *Data* object.

getWhereList(*condition*, *variables*, *start*, *stop*, *step*)

Run a query on a table, see [Query language](#).

Parameters

- **condition** (*string*) – The query string
- **variables** – A mapping of strings and variable values to be substituted into *condition*. This can often be left empty.
- **start** (*long*) – The index of the *first* row to consider.
- **stop** (*long*) – The index of the *last+1* row to consider.
- **step** (*long*) – The stepping interval between the *start* and *stop* rows to consider, using the same semantics as `range()`. Set to 0 to disable stepping.

Returns

A list of row indices matching the condition which can be passed as the first parameter of [readCoordinates\(\)](#) or [read\(\)](#).

Note: *variables* seems to add unnecessary complexity, should it be removed?

initialize(*columns*)

Initialize a new table. Any column values are ignored, use [addData\(\)](#) to add these values.

Parameters

columns (*Column*[]) – A list of columns whose names and types are used to setup the table.

addData(*columns*)

Append one or more full rows to the table.

Parameters

columns (*Column*[]) – A list of columns, such as those returned by [getHeaders\(\)](#), whose values are the rows to be added to the table.

update(*data*)

Modify one or more columns and/or rows in a table.

Parameters

data (*Data*) – A *Data* object previously obtained using [read\(\)](#) or [readCoordinates\(\)](#) with column values to be updated.

setMetadata(key, value)

Store additional properties associated with a Table.

Parameters

- **key** (*string*) – A key name.
- **value** (*string/int/float/long*) – The value of the property.

setAllMetadata(keyvalues)

Store multiple additional properties associated with a Table. See [setMetadata\(\)](#).

Parameters

keyvalues (*dict*) – A dictionary of key-value pairs.

getMetadata(key)

Get the value of a property.

Parameters

key (*string*) – The property name.

Returns

A property.

getAllMetadata()

Get all additional properties. See [getMetadata\(\)](#).

Returns

All key-value properties.

You may find the [Python](#) and [Java](#) annotated code samples helpful, in addition to the [examples](#) and [documentation on the API](#). These are only an introduction to using OMERO.tables and do not show its full potential, see [Going forward](#) for some inspiration.

Examples

- Hello World: [examples/OmeroTables/first.py](#)
- Creating a Measurement Table: [examples/OmeroTables/MeasurementTable.java](#)
- Querying a Table: [examples/OmeroTables/FindMeasurements.java](#)

The implementation

Currently, each table is backed by a single HDF table. Since PyTables (and HDF in the general case) do not support concurrent access, OMERO.tables provides a global locking mechanism which permits multiple views of the same data. Each *OMERO.tables* file (registered as an *OriginalFile* in the database), is composed of a single HDF table with any number of certain limited column types.

Query language

The query language mentioned above is *currently* the PyTables [Condition syntax](#). Columns are referenced by name. The following operators are supported:

- Logical operators: `&`, `|`, `~`
- Comparison operators: `<`, `<=`, `==`, `!=`, `>=`, `>`
- Unary arithmetic operators: `-`
- Binary arithmetic operators: `+`, `-`, `*`, `/`, `**`, `%`

and the following functions:

- `where(bool, number1, number2)`: `number` — `number1` if the `bool` condition is true, `number2` otherwise.
- `{sin,cos,tan}(float|complex)`: `float|complex` — trigonometric sine, cosine or tangent.
- `{arcsin,arccos,arctan}(float|complex)`: `float|complex` — trigonometric inverse sine, cosine or tangent.
- `arctan2(float1, float2)`: `float` — trigonometric inverse tangent of `float1/float2`.
- `{sinh,cosh,tanh}(float|complex)`: `float|complex` — hyperbolic sine, cosine or tangent.
- `{arcsinh,arccosh,arctanh}(float|complex)`: `float|complex` — hyperbolic inverse sine, cosine or tangent.
- `{log,log10,log1p}(float|complex)`: `float|complex` — natural, base-10 and $\log(1+x)$ logarithms.
- `{exp,expm1}(float|complex)`: `float|complex` — exponential and exponential minus one.
- `sqrt(float|complex)`: `float|complex` — square root.
- `{real,imag}(complex)`: `float` — real or imaginary part of complex.
- `complex(float, float)`: `complex` — complex from real and imaginary parts.

for example, if `id` is the name of a [LongColumn](#)

```
table.getWhereList(condition='(id>5)', variables={'x':omero.rtypes.rint(5)},
    start=2, stop=10, step=3)
```

will extract a subset of rows (2, 5, 8) as indicated by `start`, `stop` and `step`, substitute 5 in place of `x` in the `condition`, and evaluate `condition` so as to return the indices of rows where column `id` is greater than 5.

Going forward

The Tables API itself provides little more than a remotely accessible store, think of it as a server for Excel-like spreadsheets. We are currently looking into the facilities that can be built on top of it, and are **very** open to suggestions. For example, the [IRoi interface](#) has been extended to filter ROIs by a given measurement. This allows seeing only those results from a particular analysis run. The following example shows how to set up such a measurement and retrieve its results:

[iroi.py](#)

For an example of production code that parses out such measurements, see [populate_roi.py](#).

The *IRoi* interface has been integrated into OMERO.insight, allowing for the visualization and export of OMERO.tables:

We are also looking into a NoSQL-style storage mechanism for OMERO, either as an alternative back-end to OMERO.tables or as an additional key-value type store. Any suggestions or ideas would be *very welcome*.

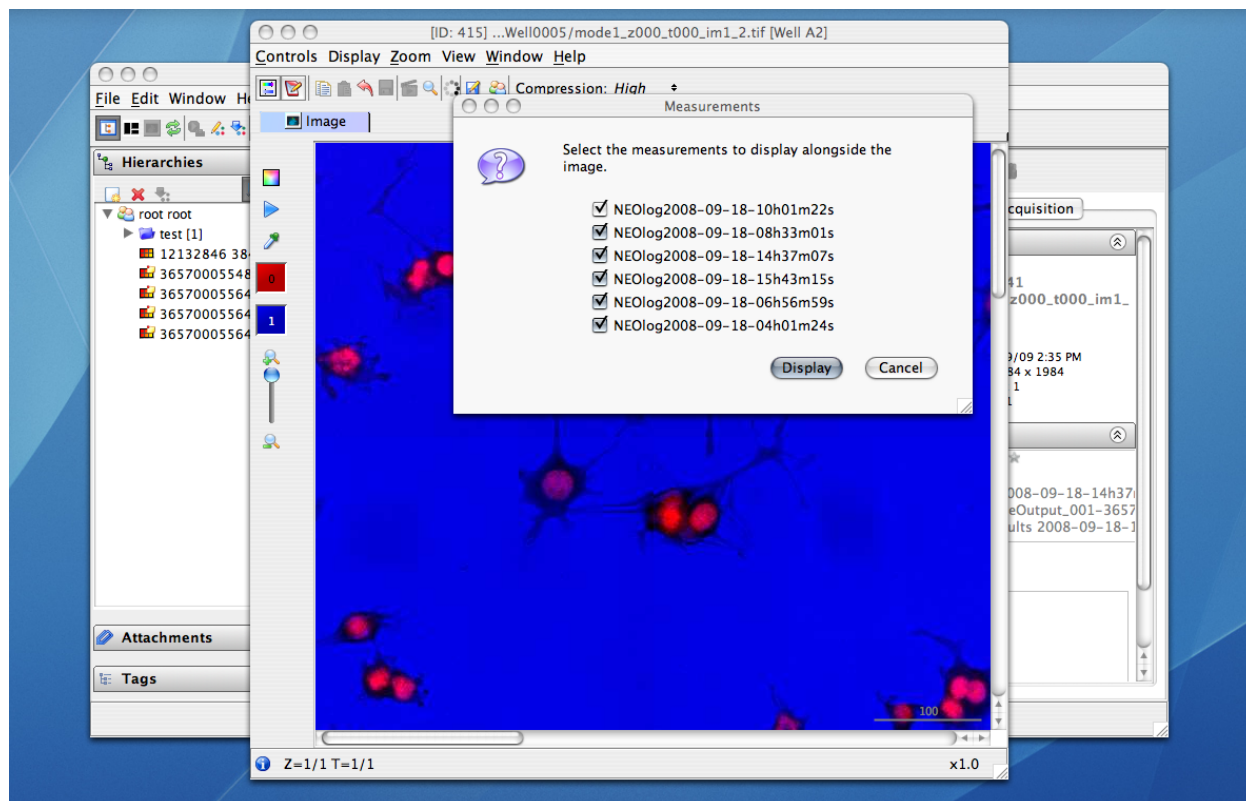


Fig. 1: Choice between multiple measurements

See also:

PyTables

Software on which OMERO.tables is built.

Condition Syntax

The PyTables condition syntax.

slice definition file

The API definition for OMERO.tables

The Tables test suite

The testsuite for OMERO.tables

3.4 Scripts - plugins for OMERO

3.4.1 Introduction to OMERO.scripts

OMERO.scripts are the OME version of plugins, allowing you to extend the functionality of your OMERO installation.

The OMERO scripting service allows scripts to be uploaded to the server so that image processing and analysis, and other functionality, can be carried out there rather than on your local machine. Scripts are generally written in Python, but other languages are also supported, like Jython. MATLAB scripts are supported natively as well as via a Python wrapper as described in *MATLAB and Python*.

Scripts can be run from the OMERO clients, using a UI generated from the script and the results should also be

handled where relevant e.g. allowing users to view OMERO Images or Datasets created by the script, or download files or images.

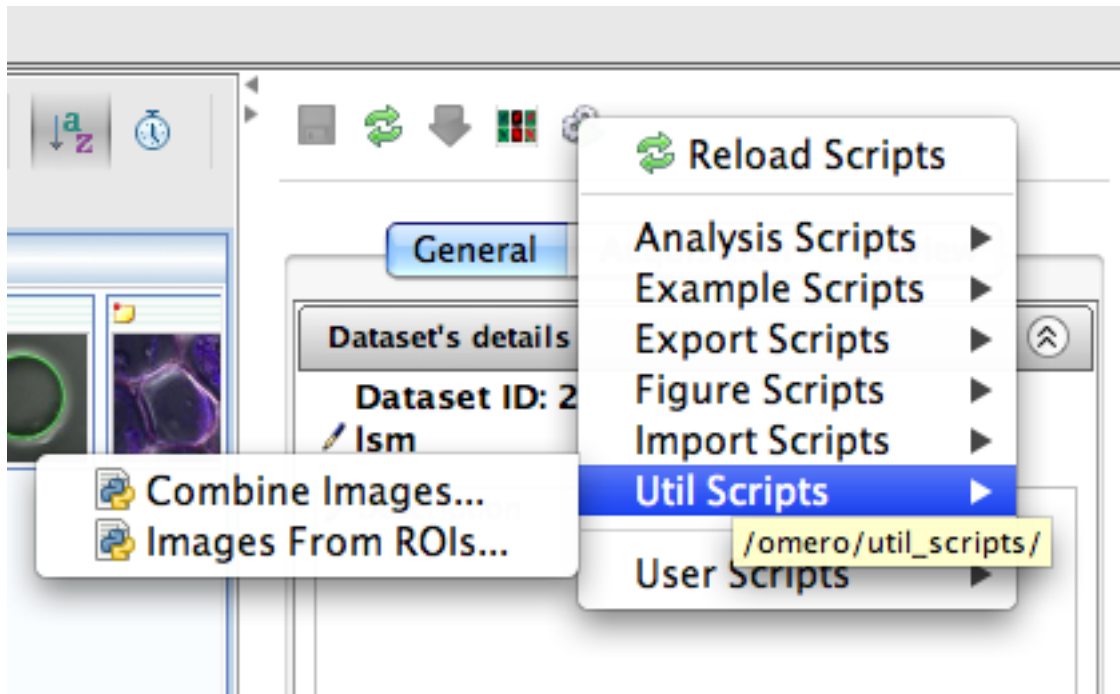


Fig. 2: Scripts menu in OMERO.insight

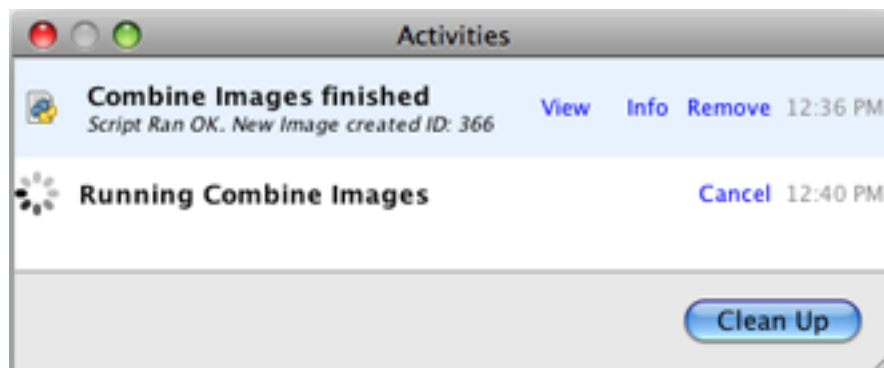


Fig. 3: Running a script from an OMERO client

Finding scripts

Core scripts are bundled with every OMERO.server release and automatically available to all users. You can find additional scripts on GitHub by looking for forks of [ome/omero-user-scripts](#). Some examples include:

- [OMERO scripts](#) - Glencoe Software
- [Example scripts](#) - OME Team
- [Fixing scripts](#) - Pierre Pouchin
- [GDSC OMERO user scripts](#) - Alex Herbert

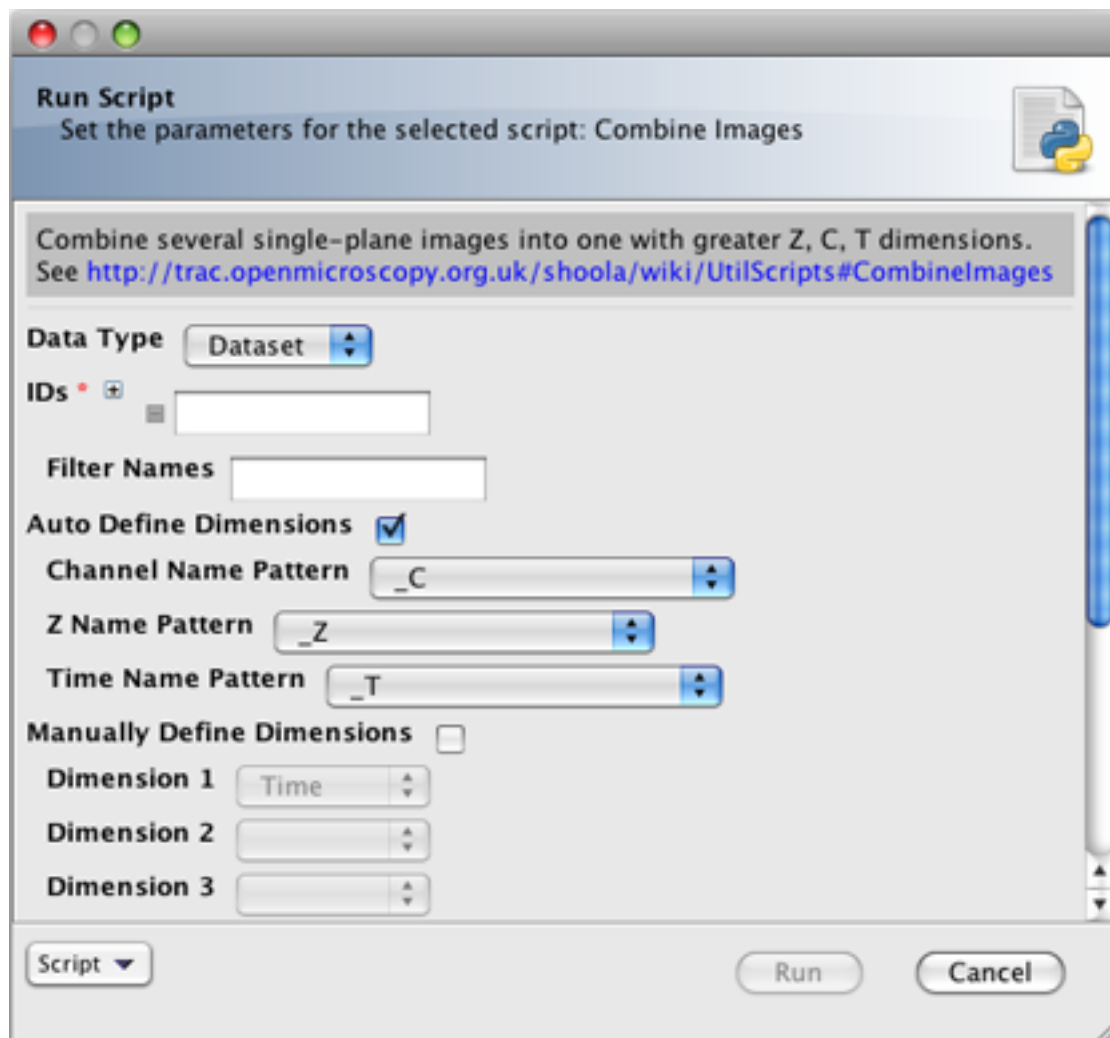


Fig. 4: A script user interface

- [QBI-Microscopy scripts](#) - Queensland Brain Institute
- [OMEROscripts](#) - Damir Sudar

All of the included scripts and repositories can be downloaded following the instructions below in order to run the scripts locally (although some of them are intended as examples only—check the associated README).

Downloading and installing scripts

The easiest way to make use of scripts is for someone with admin rights to upload them to the OMERO.server as described in the [OMERO.scripts user guide](#). Once a script has been added under the lib/scripts directory, you can run them from the OMERO clients or the command line. However, you will not be notified of any updates to the script, nor will you be able to automatically update them. This means that when your OMERO installation is upgraded, all your additional scripts will be lost.

To keep your scripts up to date, we recommend you use a Github repository to manage your scripts. If you are not familiar with [using git](#), you can use the [GitHub app for your OS](#) (available for Mac and Windows but not Linux). The basic workflow is:

- fork our [omero-user-script](#) repository or any other repository you trust (<https://github.com/omero-user-scripts/network/members>)
- clone it in your lib/scripts directory

```
cd lib/scripts
git clone git@github.com:YOURGITUSERNAME/omero-user-scripts.git YOUR_SCRIPTS
```

- save the scripts you want to use into the appropriate sub-directory in your cloned location lib/scripts/YOUR_SCRIPTS

If all you want to do is add scripts from someone else's repository to your server, you can simply clone that repository in your lib/scripts directory and the scripts within it will be added to your script list as described in the [OMERO-user-script repository readme](#).

As your new scripts will then show up in the script menu in the clients, alongside the core 'omero' scripts which are shipped with each release, you should try to pick unique names to avoid future clashes e.g. Custom_Scripts/Search_Scripts/original_metadata_search.py:

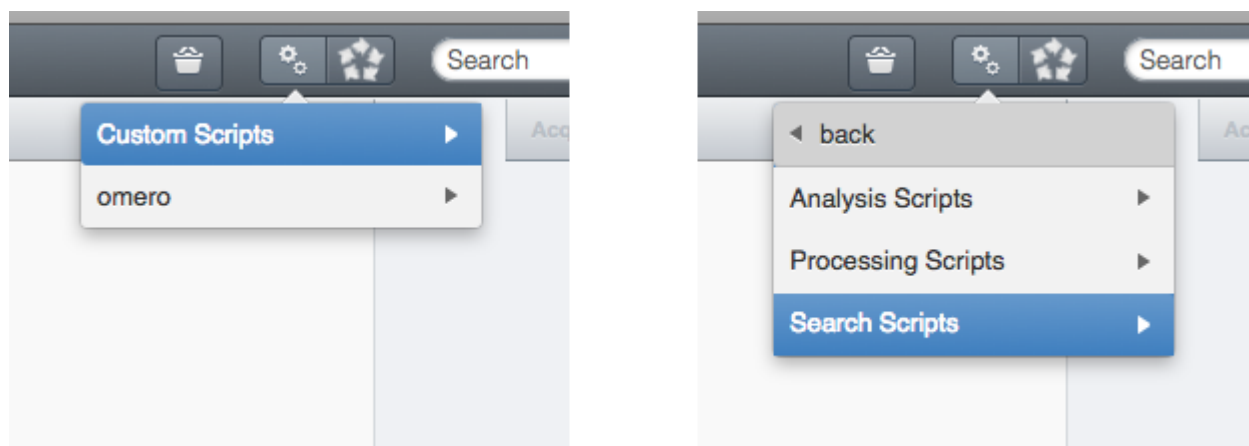


Fig. 5: Custom scripts in OMERO.web menu

The OME developers use Github to co-ordinate all our development work so joining the network will help you access help and support, and see what other people are doing with scripts. Cloning our repository also means you have an example script to get you started with developing your own.

Developing your own scripts

The easiest way to get started developing scripts for your own site is to fork the github.com/ome/omero-user-scripts repository and clone it somewhere under lib/scripts as described above. Then go into YOUR_SCRIPTS and rename the existing script to match your needs:

```
cd lib/scripts/YOUR_SCRIPTS
git mv Example.py util_scripts/New_function.py
```

Once you have done that, you can edit and test run the script and then when you are happy with it, you can save it and push it back to your fork for others to see and share.

OMERO.scripts user guide describes the workflows for developing and running your own scripts. You should use the *Guidelines for writing OMERO.scripts* to ensure your script interacts with the OMERO clients in a usable way.

Contributing back to the community

If you have modified one of the core scripts or developed your own that you would like to contribute back to the community, please get in touch. If the script is likely to have wide appeal, we can look into adding it to the core scripts that are distributed with an OMERO release.

See also:

OMERO.scripts user guide, *Guidelines for writing OMERO.scripts*, *OMERO.scripts advanced topics* and *MATLAB and Python*

3.4.2 OMERO.scripts user guide

OMERO.blitz provides a service to run scripts on the server. The scripts are then passed on to a grid of processors called OMERO.grid that executes the script and returns the result to the server which in turn passes the result onto the caller. All scripts are of the form:

```
# import the omero package and the omero.scripts package.
import omero, omero.scripts as scripts

'''
This method creates the client script object, with name SCRIPTNAME and SCRIPTDESCRIPTION.
The script then supplies a number of variables that are both inputs and outputs to the
script. The types allowed are defined in the script package, with the qualifier after the
variable of in, out or inout depending on whether the variable is for input, output or
↳input
and output.
'''
client = scripts.client("SCRIPTNAME", "SCRIPTDESCRIPTION",
    scripts.TYPE("VARIABLENAME").[in()|out()|inout()], ...)

# All variables are stored in a map accessed by getInput and setOutput via the client.
↳object.
VARIABLENAME = client.getInput("VARIABLENAME");
client.setOutput("VARIABLENAME", value);
```

This is a guide to getting started with the scripting service, without going into the ‘behind the scenes’ details. More technical details can be found on the *OMERO.scripts advanced topics* page. In addition to this guide, you may find

the following pages useful for more information on using the OMERO Python API: *Working with OMERO*, *OMERO Python language bindings*.

Sample scripts

Below are two sample scripts. You can find the core scripts that are distributed with the OMERO.server under the [scripts repository](#) or download them from OMERO.insight (from the bottom-left of any run-script dialog), or use the GitHub repositories forked from [ome/omero-user-scripts](#) to find scripts written by other users.

Ping script

This script echoes the input parameters as outputs.

```
import omero, omero.scripts as scripts
client = scripts.client("ping.py", "simple ping script",
                        scripts.Long("a"), scripts.String("b"))

keys = client.getInputKeys()
print("Keys found:")
print(keys)
for key in keys:
    client.setOutput(key, client.getInput(key))
```

Accessing an Image and Channels on the server

This example shows usage of the Python Blitz Gateway to access an Image, using its ID. We then list the Channel names and the script returns them as outputs.

```
import omero, omero.scripts as scripts
from omero.gateway import BlitzGateway
from omero.rtypes import wrap

# Define the script name & description, and a single 'required' parameter
client = scripts.client("Get_Channels.py", "Get channel names for an image",
                        scripts.Long("imageId", optional=False))

# get the Image Id from the parameters.
imageId = client.getInput("imageId", unwrap=True) # unwrap the rtype

# Use the Python Blitz Gateway for convenience
conn = BlitzGateway(client_obj=client)

# get the Image, print its name
image = conn.getObject("Image", imageId)
print(image.getName())

# Print each channel 'label' (Name or Excitation wavelength)
for i, ch in enumerate(image.getChannels()):
    print(ch.getLabel())
    # Return as output. Key is string, value is rtype
```

(continues on next page)

(continued from previous page)

```

client.setOutput("Channel%s" % i, wrap(str(ch.getLabel()))))

# Cleanup
client.closeSession()

```

Dynamic arguments

In general, script parameters are processed on server startup and cached for later use. If you have a script which should recalculate its arguments on each invocation, use the *NSDYNAMIC* namespace:

```

# A list of datasets will be dynamically generated and used to populate the
# script parameters every time the script is called

import omero
import omero.gateway
from omero import scripts
from omero.rtypes import rstring

def get_params():
    try:
        client = omero.client()
        client.createSession()
        conn = omero.gateway.BlitzGateway(client_obj=client)
        conn.SERVICE_OPTS.setOmeroGroup(-1)
        objparams = [rstring('Dataset:%d %s' % (d.id, d.getName()))
                     for d in conn.getObjects('Dataset')]
        if not objparams:
            objparams = [rstring('<No objects found>')]
        return objparams
    except Exception as e:
        return ['Exception: %s' % e]
    finally:
        client.closeSession()

def runScript():
    """
    The main entry point of the script
    """

    objparams = get_params()

    client = scripts.client(
        'Example Dynamic Test', 'Example script using dynamic parameters',

        scripts.String(
            'Dataset', optional=False, grouping='1',
            description='Select a dataset', values=objparams),

        namespaces=[omero.constants.namespaces.NSDYNAMIC],

```

(continues on next page)

(continued from previous page)

```

)

try:
    scriptParams = client.getInputs(unwrap=True)
    message = 'Params: %s\n' % scriptParams
    print(message)
    client.setOutput('Message', rstring(str(message)))

finally:
    client.closeSession()

if __name__ == '__main__':
    runScript()

```

Example_Dynamic_Script.py

Script writing as ‘Admin’

The basic steps in a script-writing workflow are:

- Write a script using your favorite text editor, save locally
- Use command line (or OMERO.insight) to upload the script to server
- Use command line (or OMERO.insight or web clients) to run the script on the server (results will be displayed)
- Edit the script and replace it on the server and run again, etc.

Working with scripts is far more straightforward if you have admin access to your OMERO.server installation - this is the preferred workflow. It is possible to work with scripts as a regular user (see *OMERO.scripts advanced topics*) but the software you would be required to install means it is easier to install a server on your local machine so you have admin rights.

It is assumed that scripts written by a server admin are “trusted” to run on the server without causing any disruption or security risks. Once uploaded these scripts are available to all regular users of the server alongside the official scripts included in each OMERO release.

Download / Edit script

The easiest way to get started is to take an existing script and edit it for your needs. An example created for the purpose of this tutorial can be found at [Edit_Descriptions.py](#). You should organize your scripts on your local machine in a way that makes sense to users, since your local file paths will be mimicked on the server and used to organize script menus in the clients (see screen-shot above).

```

# Save the script to a suitable location. The tutorial will use this location:
# Desktop/scripts/demo_tutorial/Edit_Descriptions.py

```

The action of this script (editing Image descriptions) is trivial but it demonstrates a number of features that you may find useful, including conventions for inputs and outputs to improve interaction with the clients(as discussed on the *Guidelines for writing OMERO.scripts*).

The script is well-documented and should get you started. A few points to note:

If you are using the ‘Blitz Gateway’ then you can get a connection wrapper like this:

```
from omero.gateway import BlitzGateway

conn = BlitzGateway(client_obj=client)
# now you can do e.g. conn.getObject("Image", imageId) etc.
```

Alternatively, if you are working directly with the OMERO services, you can get a service factory like this:

```
session = client.getSession()
# now you can do e.g. session.getQueryService() etc.
```

More example scripts

Several official scripts are included in the release of OMERO and can be found under the `lib/scripts/omero/` directory of the server package. Any script can also be download from the clients (bottom-left of the run-script dialog).

Warning: If you wish to edit the official scripts that are part of the OMERO release, you should be prepared to apply the same changes to future releases of these scripts from OMERO. If you think that your changes should be included as part of future released scripts, please let us know.

Upload script

You can use the command line, OMERO.insight or the server file system to upload scripts. The script command line tool is discussed in more detail below.

Upload the script we saved earlier, specifying it as ‘official’ (trusted to run on the server processor). You will need to log in the first time you use the **omero script** command. The new script ID will be printed out:

```
$ cd Desktop/scripts/
$ omero script upload demo_tutorial/Edit_Descriptions.py --official
Previously logged in to localhost:4064 as root
Server: [localhost]          # hit 'enter' to accept default login details
Username: [root]
Password:
Created session 09fcf689-cc85-409d-91ac-f9865dbfd650 (root@localhost:4064). Idle_
↪ timeout: 10.0 min. Current group: system
Uploaded official script as original file #301
```

You can add, remove and edit scripts directly in the `OMERO_HOME/lib/scripts/omero/` folder. Any changes made here will be detected by OMERO. Official scripts are uniquely identified on the OMERO server by their ‘path’ and ‘name’.

Any folders in the script path are created on the server under `/lib/scripts/` e.g. the above example will be stored at `/lib/scripts/examples/Edit_Descriptions.py`

The ID of the script is printed after upload and can also be determined by listing scripts (see below).

Run script

You can run the script from OMERO.insight or OMERO.web by browsing the scripts (see screen-shot above). A UI will be generated from the chosen script and the currently selected images or datasets will be populated if the script supports this (see [Guidelines for writing OMERO.scripts](#)).

Or launch the script from the command line, specifying the script ID. You will be asked to provide input for any non-optional parameters that do not have default values specified. Any stdout and stderr will be displayed as well as any outputs that the script has returned.

```
$ omero script launch 301 # script ID
Using session 1202acc0-4424-4fa2-84fe-7c9e069d3563 (root@localhost:4064). Idle timeout:
↳ 10.0 min. Current group: system
Enter value for "IDs": 1201
Job 1464 ready
Waiting...
Callback received: FINISHED

*** start stdout ***
* {'IDs': [1201L], 'Data_Type': 'Dataset'}
* Processing Images from Dataset: LSM - .mdb
* Editing images with this description:
* No description specified
*
*   Editing image ID: 15651 Name: sample files.mdb [XY-ch-02]
*   Editing image ID: 15652 Name: sample files.mdb [XY-ch-03]
*   Editing image ID: 15653 Name: sample files.mdb [XY-ch]
*   Editing image ID: 15654 Name: sample files.mdb [XYT]
*   Editing image ID: 15655 Name: sample files.mdb [XYZ-ch-20x]
*   Editing image ID: 15656 Name: sample files.mdb [XYZ-ch-zoom]
*   Editing image ID: 15658 Name: sample files.mdb [XYZ-ch0]
*   Editing image ID: 15657 Name: sample files.mdb [XYZ-ch]
*
*** end stdout ***

*** out parameters ***
* Message=8 Images edited
*** done ***
```

Parameter values can also be specified in the command.

```
# simply specify the required parameters that don't have defaults
$ omero script launch 301 IDs=1201

# can also specify additional parameters
$ omero script launch 301 Data_Type='Image' IDs=15652,15653 New_Description="Adding
↳ description from script to Image"
```

Edit and replace

Edit the script and upload it to replace the previous copy, specifying the ID of the file to replace.

```
$ omero script replace 301 examples/Edit_Descriptions.py
```

Finally, you can upload and run your scripts from OMERO.insight.

Other script commands

Start by printing help for the script command:

```
$ omero script -h
usage: omero script
       [-h] <subcommand> ...

Support for launching, uploading and otherwise managing OMERO.scripts

Optional Arguments:
  In addition to any higher level options

  -h, --help            show this help message and exit

Subcommands:
  Use omero script <subcommand> -h for more information.

<subcommand>
  demo                  Runs a short demo of the scripting system
  list                  List files for user or group
  cat                   Prints a script to standard out
  edit                  Opens a script in $EDITOR and saves it back to the server
  params                Print the parameters for a given script
  launch                Launch a script with parameters
  disable               Makes script non-executable by setting the mimetype
  enable                Makes a script executable (sets mimetype to text/x-python)
  jobs                  List current jobs for user or group
  serve                 Start a usermode processor for scripts
  upload                Upload a script
  replace               Replace an existing script with a new value
  run                   Run a script with the OMERO libraries loaded and current login
```

To list scripts on the server:

```
$ omero script list
Using session 09fcf689-cc85-409d-91ac-f9865dbfd650 (root@localhost:4064). Idle timeout:
↪ 10.0 min. Current group: system
id | Official scripts
-----+-----
202 | /omero/export_scripts/Batch_Image_Export.py
203 | /omero/export_scripts/Make_Movie.py
204 | /omero/figure_scripts/Movie_Figure.py
205 | /omero/figure_scripts/Movie_ROI_Figure.py
```

(continues on next page)

(continued from previous page)

```

206 | /omero/figure_scripts/ROI_Split_Figure.py
207 | /omero/figure_scripts/Split_View_Figure.py
208 | /omero/figure_scripts/Thumbnail_Figure.py
8   | /omero/import_scripts/Populate_ROI.py
209 | /omero/util_scripts/Channel_Offsets.py
210 | /omero/util_scripts/Combine_Images.py
211 | /omero/util_scripts/Images_From_ROIs.py
(14 rows)

```

If you want to know the parameters for a particular script you can use the `params` command. This prints out the details of the script, including the inputs.

```

$ omero script params 301
Using session 1202acc0-4424-4fa2-84fe-7c9e069d3563 (root@localhost:4064). Idle timeout:
↳ 10.0 min. Current group: system

id: 301
name: Edit_Descriptions.py
version:
authors:
institutions:
description: Edits the descriptions of multiple Images,
either specified via Image IDs or by the Dataset IDs.
namespaces:
stdout: text/plain
stderr: text/plain
inputs:
  New_Description - The new description to set for each Image in the Dataset
    Optional: True
    Type: ::omero::RString
    Min:
    Max:
    Values:
  IDs - List of Dataset IDs or Image IDs
    Optional: False
    Type: ::omero::RList
    Subtype: ::omero::RLong
    Min:
    Max:
    Values:
  Data_Type - The data you want to work with.
    Optional: False
    Type: ::omero::RString
    Min:
    Max:
    Values: Dataset, Image
outputs:

```

Debugging scripts

The stderr and stdout from running a script should always be returned to you, either when running scripts from the command line, via the clients or using the scripts API. This should allow you to debug any problems you have.

You can also look at the output from the script in the OriginalFile directory, commonly stored in /OMERO/File/. The script file when executed is uploaded as a new OriginalFile, and the standard error, standard out are saved as the next two OriginalFiles after that. These files can be opened in a text editor to examine contents.

3.4.3 Guidelines for writing OMERO.scripts

These guidelines for writing OMERO scripts are designed to improve the interaction of the scripts with OMERO clients so that they can:

- generate a nice, usable UI for the script
- handle the script results appropriately

If you want instructions on how to get started with OMERO scripts, see the link above or the *OMERO.scripts user guide*.

Most of the points below are implemented in the example `Edit_Descriptions.py`.

Script naming and file path

- Script Name should be in the form 'Script_Name.py'. The OMERO.web and OMERO.insight clients will replace underscores with spaces in the script selection menu.
- File paths - The clients will use the parent folder to build a scripts menu, capitalising and removing underscores. For example, a script uploaded from /omero/export_scripts/Batch_Image_Export.py will be displayed in the clients under "Export Scripts".
- Script Descriptions should give a brief summary of what the script does. If a longer description or instructions for using the script are desired, it is suggested that a URL is included. The description will be displayed in the script UI and any URLs will be 'clickable' to launch a browser.

Parameters

- Parameter Names should be in the form 'Parameter_Name'. Underscores will be replaced by spaces in the UI generated in the clients.
- Where applicable, parameters should be supplied with a list of options. For example:

```
scripts.String("Algorithm", values=[rstring('Maximum_Intensity'),rstring('Mean_
↪Intensity')] )
```

- Where possible, parameters should be supplied with default values. These will be used to populate fields in the clients script UI and will be used by default when launching the script from the command line.

```
scripts.String("Folder_Name", description="Name of folder to store images", default=
↪'Batch_Image_Export'),
```

- Where applicable, parameters should have min and max values, e.g.:

```
scripts.Int("Size_Z", description="Number of Z planes in new image", min=1),
```

Run Script
Set the parameters of the selected script: Movie ROI Figure

Create a figure of movie frames from ROI region of image.
See <http://trac.openmicroscopy.org.uk/shoola/wiki/FigureExport#ROI-Movie-Figure>
Authors: William Moore, OME Team
Contact: ome-users@lists.openmicroscopy.org.uk
Version: 4.2.0

Data Type Image

IDs *

Merged Colours Blue

Merged Channels

Roi Zoom 1.0

Max Columns 10 Min: 1

Resize Images ☒

Height Min: 1

Width Min: 1

Image Labels Image Name

Show ROI Duration ☐

Scalebar Min: 1

Scalebar Colour White

Roi Selection Label

Algorithm Maximum Intensity

Figure Name movieROIfigure

Format JPEG

* indicates the required parameter.

Script Run Cancel

Parameter grouping / ordering

Parameters are not ordered by default. They can be ordered and grouped by adding a “grouping” attribute, which is a string, where ‘groups’ are separated by a ‘.’ e.g. “01.A”. Parameters will be ordered by the lexicographic sorting of this string and groups indicated in the UI. In most cases this will simply be a common indentation of parameters in the same group. In addition, if the ‘parent’ parameter of a group is a boolean, then un-checking the check-box in the UI will disable the child parameters. For example a UI generated from the code below will have a ‘Show Scalebar’ option. If this is un-checked, then the ‘Size’ and ‘Colour’ parameters will be disabled and will not be passed to the script.

```
scripts.Bool("Show_Scalebar", grouping="10", default=True),
scripts.Int("Scalebar_Size", grouping="10.1"),
scripts.String("Scalebar_Colour", grouping="10.2"),
```

Pick selected Images, Datasets or Projects from OMERO clients

Both OMERO.insight and OMERO.web recognize and populate a pair of fields named ‘Data_Type’ (string) and ‘IDs’ (Long list) with the objects currently selected in the client UI when the script is launched. You should specify the ‘Data_Type’ options that your script should accept. For example:

```
dataTypes = [rstring('Dataset'),rstring('Image')]

client = scripts.client('Thumbnail_Figure.py', "Export a figure of thumbnails",
    scripts.String("Data_Type", optional=False, grouping="01", values=dataTypes, default=
    ↪ "Dataset"),
    scripts.List("IDs", optional=False, grouping="02").ofType(rlong(0))
)
```

Script outputs

- Scripts may return a short message to report success or failure. This should use the key: ‘Message’ in the output map. This will be displayed in clients when the script completes.

```
client.setOutput("Message", rstring("Script generated new Image"))
```

- Scripts that generate an Image should return the omero.model.ImageI object. The clients will provide a link to view the Image. The key that is used (“Image” in this example) is not important for this to work, but ‘image’ should be an omero.model.ImageI object.

```
client.setOutput("Image",robject(image))
```

- Scripts that generate a File Annotation or Original File should return these objects. The clients will give users the option of downloading the File, and may also allow viewing of the file if it is of a suitable type. This should be set as the mimetype of the File Annotation (e.g. ‘plain/text’, ‘image/jpeg’, etc.). In this example, fileAnnotation should be an omero.model.FileAnnotationI object, but could also be an omero.model.OriginalFileI object.

```
client.setOutput("File_Annotation",robject(fileAnnotation))
```

- Scripts that generate a URL link should return the omero.rtypes.rmap, with the following keys: “type”: “URL”, “href”: “URL address to open”, “title”: “Help message”. The client will give users the option of opening the URL in a new browser window/tab. To use this feature the URL omero.rtypes.rmap should use the key: ‘URL’ in the output map.

```
url = omero.rtypes.wrap({
    "type": "URL",
    "href": "https://www.openmicroscopy.org",
    "title": "Open URL link to OME's website.",
})
client.setOutput("URL", url)
```

More tips

- Use the 'unwrap()' function from omero.rtypes to unwrap rtypes from the script parameters since this function will iteratively unwrap lists, maps, etc..

```
from omero.rtypes import *
scriptParams = {}
for key in client.getInputKeys():
    if client.getInput(key):
        scriptParams[key] = unwrap(client.getInput(key))

print(scriptParams)    # stdout will be returned - useful for bug fixing etc.
```

3.4.4 MATLAB and Python

MATLAB functionality can be mixed into Python scripts using the MATLAB Engine API for Python.

Installing MATLAB Engine API

To install the MATLAB Engine API for Python follow the [installation guide](#). You only need to run `python setup.py install`, most likely as an administrator. It is possible to install the engine into a virtual environment.

Example MATLAB scripts

Below are some sample scripts showing MATLAB being launched from OMERO.scripts. MATLAB functions can also call the *OMERO Java language bindings* interface to access the server from the MATLAB functions.

Calling a simple MATLAB function

```
import omero.scripts as scripts

import matlab.engine

client = scripts.client('prime.py',
    """
    This script checks if the specified number is a prime number.
    """,
    scripts.Long(
        "x", optional=False, grouping="1",
        description="Number to check.))
try:
```

(continues on next page)

(continued from previous page)

```

# process the list of args above.
params = {}
for key in client.getInputKeys():
    if client.getInput(key):
        params[key] = client.getInput(key, unwrap=True)
x = params.get("x")
# start the MATLAB engine
eng = matlab.engine.start_matlab("-nodisplay")
tf = eng.isprime(x)
print(tf)
eng.quit()
finally:
    client.closeSession()

```

Using the OMERO interface inside MATLAB

This example shows the MATLAB script being called, passed to the client object and accessing the same client instance as the script.

You will need to have the OMERO.matlab toolbox installed on the server:

- download the toolbox from the [OMERO Downloads page](#)
- unzip
- enter the full path to the toolbox in the OMERO script below.

Create a `frap.m`, copy the MATLAB function below. Save the file to the server. Enter the full path to the directory containing the script in the example OMERO script below.

Note that this script expects to run on timelapse images with at least one Ellipse not linked to a T-index.

```

import omero

import omero.scripts as scripts
from omero.rtypes import rlong
from omero.gateway import BlitzGateway
from omero.rtypes import robject, rstring

import matlab.engine

dataTypes = [rstring('Dataset')]
client = scripts.client('frap.py',
    """
This script does simple FRAP analysis using Ellipse ROIs previously
saved on images. If matplotlib is installed, data is plotted and new
OMERO images are created from the plots.
Call the matlab frap code.
    """,
    scripts.String(
        "Data_Type", optional=False, grouping="1",
        description="Choose source of images",
        values=dataTypes, default="Dataset"),

```

(continues on next page)

(continued from previous page)

```

scripts.Long(
    "ID", optional=False, grouping="2",
    description="Dataset ID.")
try:
    # process the list of args above.
    params = {}
    for key in client.getInputKeys():
        if client.getInput(key):
            params[key] = client.getInput(key, unwrap=True)
    dataset_id = params.get("ID")
    # wrap client to use the Blitz Gateway
    conn = BlitzGateway(client_obj=client)
    # start the MATLAB engine
    eng = matlab.engine.start_matlab("-nodisplay")
    # Add the OMERO.matlab toolbox the MATLABPATH
    eng.addpath("PATH_TO_TOOLBOX/OMERO.matlab-xxx")
    # Add the frap function to the MATLABPATH.
    # For convenience this could
    # be placed in the OMERO.matlab toolbox folder
    eng.addpath("PATH_TO_FRAP")
    eng.frap(conn.getEventContext().sessionId, dataset_id, nargout=0)
    eng.quit()
    client.setOutput("Message", rstring("frap script completed"))
finally:
    client.closeSession()

```

The MATLAB frap function

```

function T = frap(sessionId, datasetId)

p = inputParser;
p.addRequired('sessionId',@(x) isscalar(x));
p.addRequired('datasetId',@(x) isscalar(x));

client = loadOmero();
client.enableKeepAlive(60);
% Join an OMERO session
session = client.joinSession(sessionId);
% Initilize the service used to load the Regions of Interest (ROI)
service = session.getRoiService();

% Retrieve the Dataset with the Images
dataset = getDatasets(session, datasetId, true);
images = toMatlabList(dataset.linkedImageList);

% Iterate through the images

for i = 1 : numel(images)

```

(continues on next page)

(continued from previous page)

```

image = images(i);
imageId = image.getId().getValue();
pixels = image.getPrimaryPixels();
sizeT = pixels.getSizeT().getValue(); % The number of timepoints

% Load the ROIs linked to the Image. Only keep the Ellipses
roiResult = service.findByImage(imageId, []);
rois = roiResult.rois;
if rois.size == 0
    continue;
end
toAnalyse = java.util.ArrayList;
for thisROI = 1:rois.size
    roi = rois.get(thisROI-1);
    for ns = 1:roi.sizeOfShapes
        shape = roi.getShape(ns-1);
        if (isa(shape, 'omero.model.Ellipse'))
            toAnalyse.add(java.lang.Long(shape.getId().getValue()));
        end
    end
end

% We analyse the first z and the first channel
keys = strings(1, sizeT);
values = strings(1, sizeT);
means = zeros(1, sizeT);
for t = 0:sizeT-1
    % OMERO index starts at 0
    stats = service.getShapeStatsRestricted(toAnalyse, 0, t, [0]);
    calculated = stats(1,1);
    mean = calculated.mean(1,1);
    index = t+1;
    keys(1, index) = num2str(t);
    values(1, index) = num2str(mean);
    means(1, index) = mean;
end
% create a map annotation and link it to the Image
mapAnnotation = writeMapAnnotation(session, cellstr(keys), cellstr(values),
↳ 'namespace', 'demo.simple_frap_data');
linkAnnotation(session, mapAnnotation, 'image', imageId);

% Create a CSV
headers = 'Image_name,ImageID,Timepoint,Mean';
tmpName = [tempname, '.csv'];
[filepath, imageName, ext] = fileparts(tmpName);
f = fullfile(filepath, 'results_frap.csv');
fileID = fopen(f, 'w');
fprintf(fileID, '%s\n', headers);
for j = 1 : numel(keys)
    row = strcat(char(imageName), ',', num2str(imageId), ',', keys(1, j), ',',
↳ values(1, j));
    fprintf(fileID, '%s\n', row);

```

(continues on next page)

(continued from previous page)

```

end
fclose(fileID);
% Create a file annotation
fileAnnotation = writeFileAnnotation(session, f, 'mimetype', 'text/csv', 'namespace',
↪ 'training.demo');
linkAnnotation(session, fileAnnotation, 'image', imageId);

% Plot the result
time = 1:sizeT;
fig = plot(means);
xlabel('Timepoint'), ylabel('Values');
% Save the plot as png
name = strcat(char(image.getName().getValue()), '_FRAP_plot.png');
saveas(fig, name);
% Upload the Image as an attachment
fileAnnotation = writeFileAnnotation(session, name);
linkAnnotation(session, fileAnnotation, 'image', imageId);
% Delete the local file
delete(name)

end

```

3.4.5 OMERO.scripts advanced topics

Regular user (non-admin) workflow

If you are using a server for which you do not have admin access, you must upload scripts as ‘user’ scripts, which are not trusted to run on the server machine. The OMERO scripting service will still execute these scripts in a similar manner to official ‘trusted’ scripts but behind the scenes it uses the client machine to execute the script. This means that any package imports required by the script should be available on the client machine.

The first step is to connect to the server and set up the processor on the client (see diagram, below).

- Install ‘Ice’ from ZeroC and set the environment variables, as described in the [server installation page](#).
- You also need the OMERO server download. Go to the [OMERO downloads](#) page and get the appropriate server package (version must match the server you are connecting to). Unzip the package in a suitable location.

In a command line terminal, change into the unzipped OMERO package, connect to the server and start user processor. For example for host: `openmicroscopy.org.uk` and user: `will`

```

$ cd Desktop/OMERO.server-5.3.x-icexx-bxx/
$ omero -s openmicroscopy.org.uk -u will script serve user
$ password: .....

```

You should see an output similar to the one below

```

Created session afd8ba21-35dc-462a-ab6e-15cc94f93957 (user-4@openmicroscopy.org.uk:4064).
↪ Idle timeout: 10 min. Current group: read-only-1
2016-10-03 10:12:45,964 INFO [ omero.util.Resources] (Thread-2 )↪
↪ Starting
2016-10-03 10:12:45,965 INFO [ omero.processor.ProcessorI] (MainThread)↪
↪ Registering processor %fOr(Up>[ERUV%B8$.N</omero.scripts.serve-fa53ba-3959-4d85-876a-

```

(continues on next page)

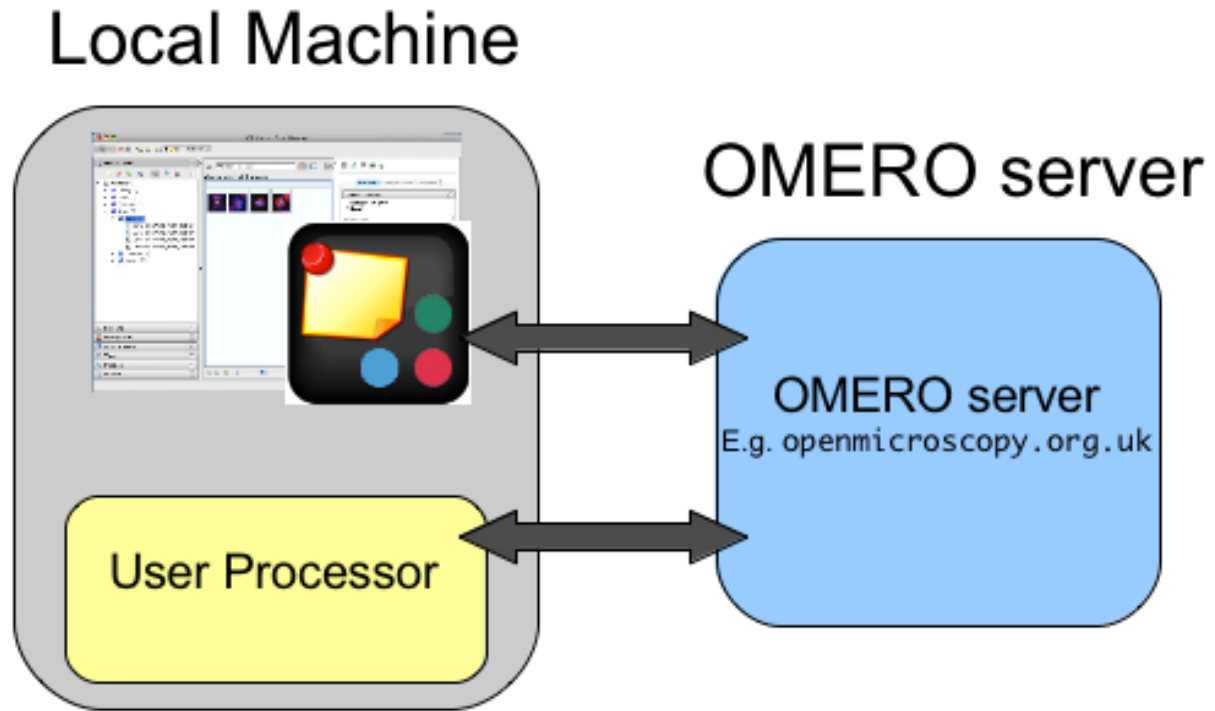


Fig. 7: OMERO scripting workflow

(continued from previous page)

```

↪ 00e8b932eb -t -e 1.0:tcp -h openmicroscopy.org.uk -p 54385
Press any key to exit...
  
```

Now you need to open a new terminal window in order to continue with your workflow.

If you want to run scripts belonging to another user in the same collaborative group you need to set up your local user processor to accept scripts from that user. First, find the ID of the user, then start the user processor and give it the user's ID:

```

$ cd Desktop/OMERO.server-5.3.x-icexx-bxx/
$ omero user list
$ omero script serve user=5
  
```

From this point on, the user and admin workflows are the same, except for a couple of options that are not available to regular users. Also see below.

Note: Because non-official scripts do not have a unique path name, you will be able to run the upload command multiple times on the same file. This will create multiple copies of a file in OMERO and then you will have to choose the most recent one (highest ID) if you want to run the latest script. It is best to avoid this and use the 'replace' command as for official scripts.

To list user scripts:

```

$ omero script list user      # lists user scripts
id | Scripts for user
  
```

(continues on next page)

(continued from previous page)

```

-----+-----
151 | examples/HelloWorld.py
251 | examples/Edit_Descriptions.py

```

You can list scripts belonging to another user that are available for you (e.g. you are both in the same collaborative group) by using the user ID as described above:

```

$ omero user list
$ omero script list user=5

```

User scripts can be run from OMERO.insight. They will be found under ‘User Scripts’ in the scripts menu. Remember, for user scripts you will need to have the User-Processor running.

The iScript service

The OMERO.blitz server provides a service called **iScript** that includes methods to upload, delete, query and run scripts. To access these methods a session needs to be created and the script service started. However, you may find it more convenient to use the command line `omero script` or the OMERO.insight client to work with scripts as described on the *OMERO.scripts user guide*.

Scripting service API

The recommended way of working with the scripting service is via the command line as described on the *OMERO.scripts user guide* page. The information on this page is only useful if you want to access the Scripting service from your own client-side Python code.

OMERO clients can upload, edit, list and run scripts on the OMERO server using the Scripting Service API.

These methods (discussed below) are implemented in <https://github.com/ome/openmicroscopy/blob/develop/examples/ScriptingService/adminWorkflow.py>. This sample script allows these functions to be called from the command line and can be used as an example for writing your own clients.

Most functions of the `adminWorkflow.py` script are also implemented in the OMERO CLI described on the *OMERO.scripts user guide*, which is the preferred way of accessing the scripting service for script writers.

Having downloaded <https://github.com/ome/openmicroscopy/blob/develop/examples/ScriptingService/adminWorkflow.py>, you can get some instructions for using the script by typing:

```

$ python adminWorkflow.py help

```

To upload ‘official’ scripts, use the `uploadOfficialScript` method of the scripting service or use the `upload` command from `adminWorkflow.py` (you can omit password and enter it later if you do not want it showing in your console):

```

$ python adminWorkflow.py -s server -u username -p password -f script/file/to/upload.py
↪upload

```

Official scripts must have unique paths. Therefore, the `uploadOfficialScript` method will not allow you to overwrite and existing script. However, the `adminWorkflow.py` `upload` command will automatically use `scriptService.editScript()` if the file exists. If you want to change this behavior, edit the `adminWorkflow.py` script accordingly.

To get the official scripts available to run, use the `getScripts()` method, which returns a list of Original Files (scripts). This code will produce a list of scripts like the one above.

```
scripts = scriptService.getScripts()
for s in scripts:
    print(s.id.val, s.path.val + s.name.val)
```

This can be called from adminWorkflow.py with this command:

```
$ python adminWorkflow.py -s server -u username -p password list
```

The script can then be run, using the script ID and passing the script a map of the input parameters. The adminWorkflow.py script has a 'run' command that can be used to identify a script by its ID or path/name and run it. The 'run' command will ask for parameter inputs at the command line.

```
$ python adminWorkflow.py -s localhost -u root -p omero -f scriptID run
```

or

```
$ python adminWorkflow.py -s localhost -u root -p omero -f omero/figure_scripts/Roi_
↪Figure.py run
```

You can combine the latter form of this command with the 'upload' option to upload and run a script at once, useful for script writing and testing.

```
$ python adminWorkflow.py -s localhost -u root -p omero -f omero/figure_scripts/Roi_
↪Figure.py upload run
```

Alternatively, you could edit adminWorkflow.py to 'hard-code' a set of input parameters for a particular script (this strategy is used by <https://github.com/ome/openmicroscopy/blob/develop/examples/ScriptingService/runHelloWorld.py>). The code below shows a more complex example parameter map. This strategy might save you time if you want to be able to rapidly run and re-run a script you are working on. Of course, it is also possible to run scripts from OMERO.insight!

```
cNamesMap = omero.rtypes.rmap({'0':omero.rtypes.rstring("DAPI"),
    '1':omero.rtypes.rstring("GFP"),
    '2':omero.rtypes.rstring("Red"),
    '3':omero.rtypes.rstring("ACA")})
blue = omero.rtypes.rstring('Blue')
red = omero.rtypes.rstring('Red')
mrgdColoursMap = omero.rtypes.rmap({'0':blue, '1':blue, '3':red})
map = {
    "Image_IDs": omero.rtypes.rlist(imageIds),
    "Channel_Names": cNamesMap,
    "Split_Indexes": omero.rtypes.rlist([omero.rtypes.rlong(1),omero.rtypes.rlong(2)]),
    "Split_Panels_Grey": omero.rtypes.rbool(True),
    "Merged_Colours": mrgdColoursMap,
    "Merged_Names": omero.rtypes.rbool(True),
    "Width": omero.rtypes.rint(200),
    "Height": omero.rtypes.rint(200),
    "Image_Labels": omero.rtypes.rstring("Datasets"),
    "Algorithm": omero.rtypes.rstring("Mean_Intensity"),
    "Stepping": omero.rtypes.rint(1),
    "Scalebar": omero.rtypes.rint(10), # will be ignored since no pixelsize set
    "Format": omero.rtypes.rstring("PNG"),
    "Figure_Name": omero.rtypes.rstring("splitViewTest"),
    "Overlay_Colour": omero.rtypes.rstring("Red"),
```

(continues on next page)

(continued from previous page)

```

"ROI_Zoom":omero.rtypes.rfloat(3),
"ROI_Label":omero.rtypes.rstring("fakeTest"), # won't be found - but should still work
}

```

The results returned from running the script can be queried for script outputs, including stdout and stderr. The following code queries the results for an output named 'Message' (also displayed by OMERO.insight)

```

print(results.keys())
if 'Message' in results:
    print(results['Message'].getValue())
if 'stdout' in results:
    origFile = results['stdout'].getValue()
    print("Script generated StdOut in file:" , origFile.getId().getValue())
if 'stderr' in results:
    origFile = results['stderr'].getValue()
    print("Script generated StdErr in file:" , origFile.getId().getValue())

```

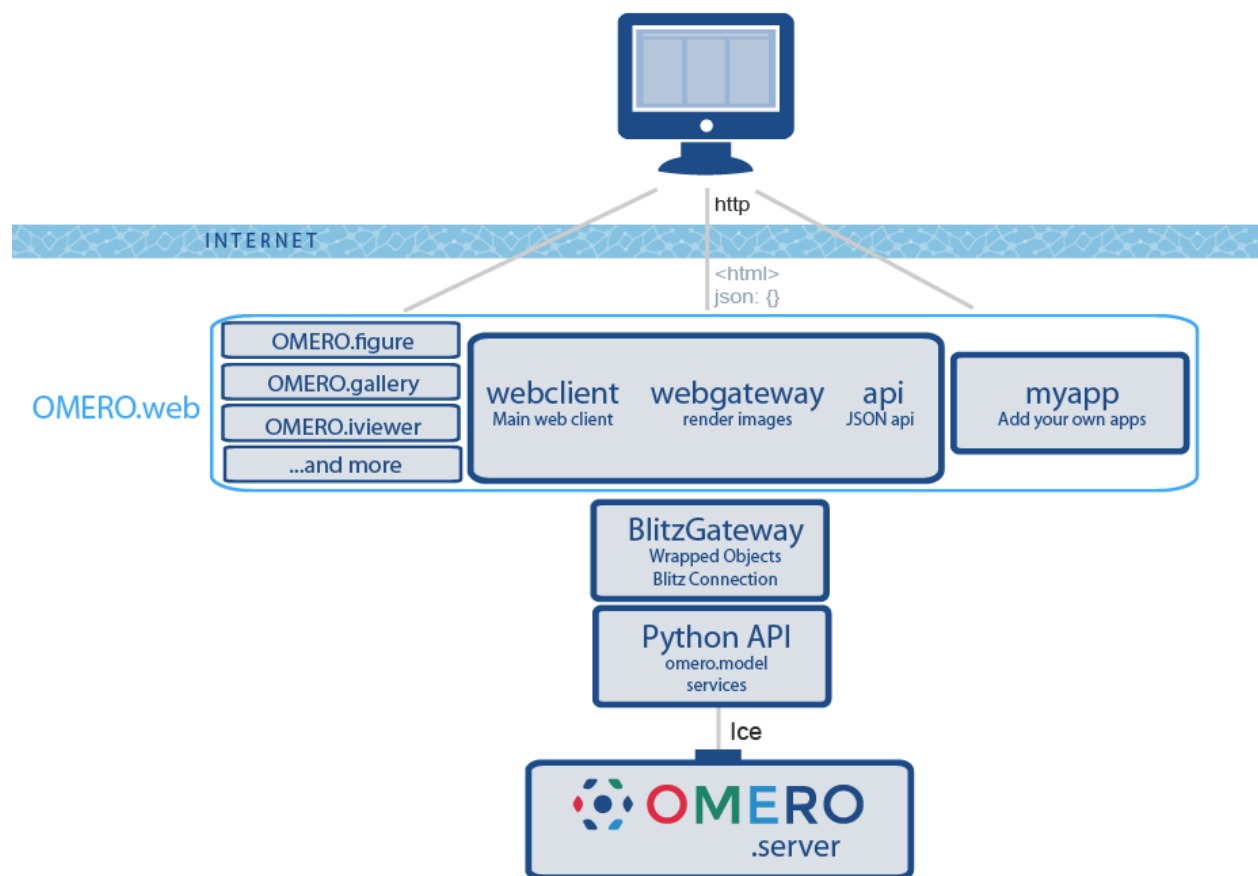
This code has been extended in adminWorkflow.py to display any StdErr and StdOut generated by the script when it is run.

3.5 Web

3.5.1 OMERO.web framework

OMERO.web is a framework for building web applications for OMERO. It uses [Django](#) to generate HTML and JSON from data retrieved from the OMERO server. OMERO.web acts as a Python client of the OMERO server using the OMERO API, as well as being a web server itself (see 'infrastructure' info below). It uses Django 'apps' to provide modular web tools, such as the webclient, webgateway and the JSON api app. This modular framework makes it possible to extend OMERO.web with your own apps.

Since version 5.14.0, OMERO.web depends on Django 3.2.x. Plugin developers should read the [Guide](#) detailing how to migrate their plugin(s) to Django 3.2.x.



OMERO.web infrastructure

OMERO Python API

The OMERO.web framework is all based on the OMERO Python API. Code-generated omero.model objects communicate remotely with their counterparts on the OMERO.server using [Ice from ZeroC](#).

BlitzGateway

The Blitz Gateway wraps omero.model objects to facilitate many loading and update operations (see [Blitz Gateway documentation](#)).

OMERO.web

The OMERO.web framework consists of several Django apps that are included in the OMERO.server release, as well as others that can be installed independently (see below). It also includes utilities for creating and retrieving connections to OMERO (see example below and [Writing OMERO.web views](#) for more details).

Included apps

- **webclient:** Main web client for browsing, viewing and annotating images. More information available under [OMERO.web](#).
- **webgateway:** Provides rendered images and JSON data for other OMERO.web apps or for external applications hosted elsewhere. See [WebGateway](#).
- **webadmin:** Tool for OMERO.server Administrators to manage users and groups.
- **api:** New in 5.3.0, this provides a JSON API for OMERO. See [JSON API](#).

Additional apps

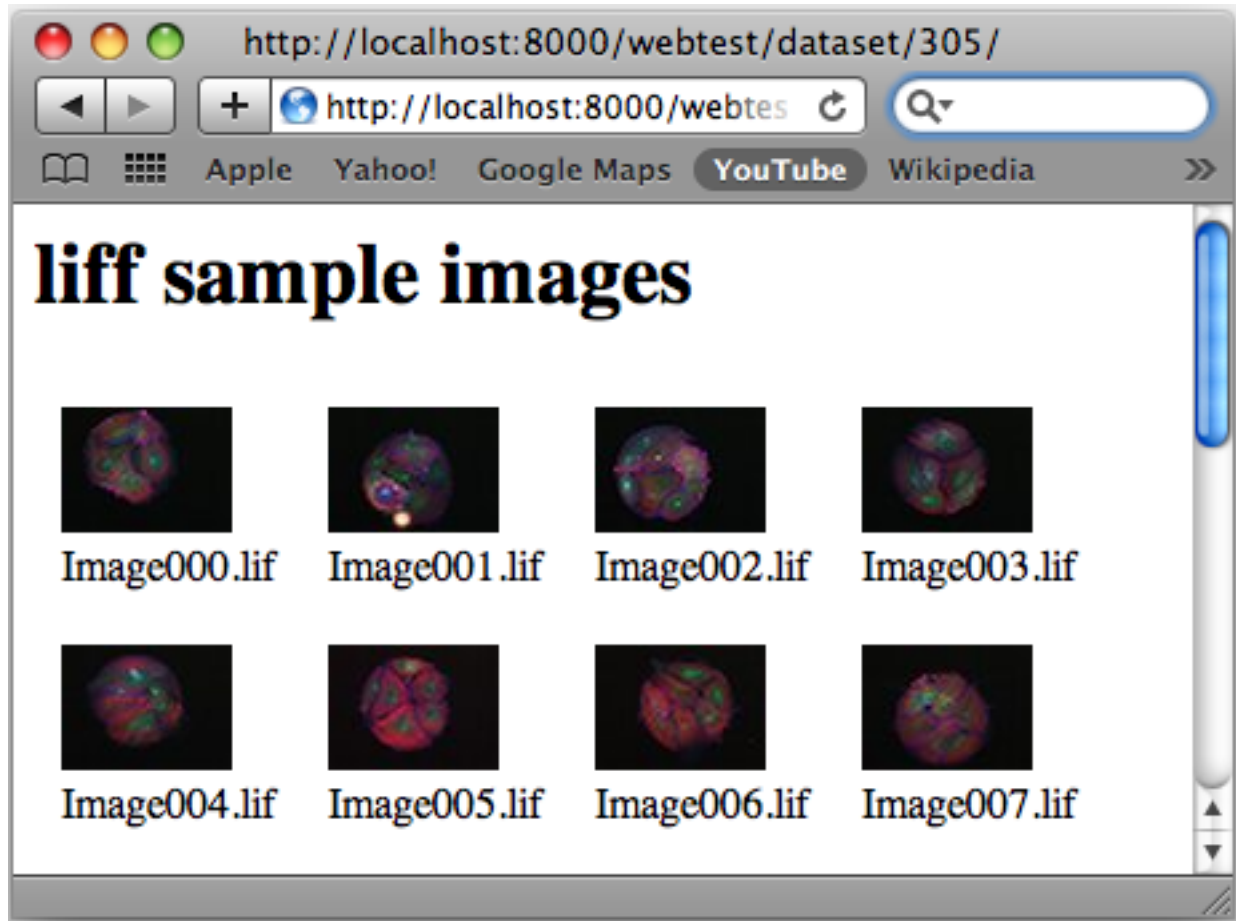
- **omero-figure:** The [OMERO.figure](#) app allows you to create scientific figures.
- **omero-webtest:** [webtest](#) is an example app that contains several code samples mentioned in the following pages.
- **omero-iviewer:** The [OMERO.iviewer](#) is a new (currently unreleased) image viewer that supports ROI creation and editing.
- **webtagging:** The webtagging apps [OMERO.autotag](#) and [OMERO.tagsearch](#) were developed externally by Douglas Russell. It supports 'auto' tagging based on image name and Tag-based filtering of data.
- **omero-mapr:** The [OMERO.mapr](#) app is a new tool that allows browsing data through Map Annotations linked to images.
- **omero-gallery:** [OMERO.gallery](#) provides a simple interface for browsing Projects, Datasets and Images.

Warning: Although it is possible to access functionality from any installed app, **ONLY webgateway and api** should be considered as a stable public API. URLs and methods within other web apps are purely designed to provide internal functionality for the app itself and may change in minor releases without warning.

Getting started

The preferred workflow for extending OMERO.web is to create a new Django app. Django apps provide a nice way for you to keep all your code in one place and make it much easier to port your app to new OMERO releases or share it with other users. To get started, see [Creating an app](#). Further documentation on editing the core OMERO.web code is at [Editing OMERO.web](#). If you want to have a quick look at some example code, see below.

Quick example - OMERO.webtest



This tiny example gives you a feel for how the OMERO.web framework gets data from OMERO and displays it on a web page. You can find this and other examples in the [OMERO.webtest](#) repository.

There are 3 parts to each page: url, view and template. For example, this code below is for generating an HTML page of a Dataset (see screen-shot). If you have OMERO.web running and webtest installed, you can view the page under `http://<servername>/webtest/dataset/<datasetId>`.

- **url** goes in `omeroweb/omero_webtest/urls.py` This maps the URL `'webtest/dataset/<datasetId>/'` to the View function `'dataset'`, passing it the `datasetId`.

```
url( r'^dataset/(?P<dataset_id>[0-9]+)/$', views.dataset, name="webtest_dataset" ),
```

- **view** function, in `omeroweb/omero_webtest/views.py`. N.B.: `@login_required` decorator retrieves connection to OMERO as `'conn'` passed in args to method. See [Writing OMERO.web views](#) for more details.


```

from omeroweb.webclient.decorators import login_required
# handles login (or redirects)
@login_required()
def dataset(request, dataset_id, conn=None, **kwargs):
    ds = conn.getObject("Dataset", dataset_id)
    # generate html from template
    return render(request, 'webtest/dataset.html', {'dataset': ds})

```

- **template:** The template web page, in omero-webtest/omero_webtest/templates/webtest/dataset.html

```

<html><body>

<h1>{{ dataset.getName }}</h1>

{% for i in dataset.listChildren %}
    <div style="float:left; padding:10px">
        
        <br />
        {{ i.getName }}
    </div>
{% endfor %}

</body></html>

```

- Next: Get started by [OMERO.web installation for developers...](#)

3.5.2 OMERO.web installation for developers

Getting set up

You will need to have an OMERO server running that you can connect to. This could be on your own machine localhost or you can connect to a remote OMERO server.

The preferred option for developing OMERO.web apps is to install omero-web on your machine as described below.

However, it is also possible to use [omero-web-docker](#) to run OMERO.web in a Docker container. If you are using this option, you can go directly to the [Creating an app](#) page which describes this process.

Installing OMERO.web

From OMERO 5.6.0 release, the omero-web library supports Python 3 and can be installed via **pip**. We need to specify a location OMERODIR to create log files and a config.xml file. This can be any existing directory. We recommend you use a virtual environment:

```

$ python3 -m venv py3_venv
$ source py3_venv/bin/activate

```

```

# Somewhere to create config and log files
$ export OMERODIR=$(pwd)

```

```

$ pip install 'omero-web>=5.24.0'

```

Using the lightweight development server

All that is required to use the Django lightweight development server is to set the `omero.web.application_server` configuration option, and turn `omero.web.debug` on. If you want to connect to a remote OMERO server, add that as shown. Then start up the development server to run in the foreground:

```
$ omero config set omero.web.application_server development
$ omero config set omero.web.debug True
$ omero config append omero.web.server_list '["server.address", 4064, "name"]'
$ omero web start
INFO:__main__:Application Starting...
INFO:root:Processing custom settings for module omeroweb.settings
...
Validating models...

0 errors found
Django version 3.2, using settings 'omeroweb.settings'
Starting development server at http://127.0.0.1:4080/
Quit the server with CONTROL-C.
```

You should now be able to open <http://localhost:4080> in your browser, choose the OMERO server and login.

Using WSGI

For convenience you may wish to run a web server under your local user account instead of using a system server for testing. Install NGINX and Gunicorn (See [OMERO.web installation and maintenance](#)) but generate a configuration file using the following commands:

```
$ omero config set omero.web.application_server 'wsgi-tcp'
$ omero web config nginx-development > nginx-development.conf
```

Start NGINX and the Gunicorn worker processes running one thread listening on 127.0.0.1:4080 that will autoreload on source change:

```
$ nginx -c $PWD/nginx-development.conf
$ omero config set omero.web.application_server.max_requests 1
$ omero config set omero.web.wsgi_args -- "--reload"
$ omero web start
```

Next: Get started by [Creating an app](#)...

3.5.3 Creating an app

The Django web site has a very good [tutorial](#) to get you familiar with the Django framework. The more you know about Django, the easier you will find it working with the OmeroWeb framework.

All official OMERO applications can be installed from [PyPI](#).

Getting set up

In order to deploy OMERO.web in a development or testing environment, you can use Docker as described below or follow the install instructions under *OMERO.web installation for developers*.

If you want to make changes to the OMERO.web code itself, go to *Editing OMERO.web*.

Clone the examples repository

To get started quickly, we are going to use the `omero web apps examples` repository which contains two sample OMERO.web apps. Clone the repo to a location of your choice:

```
$ git clone git@github.com:ome/omero-web-apps-examples.git
```

We will run the simplest example app from the repo. This is called `minimal_webapp`.

Run your app with locally-installed OMERO.web

If you have installed `omero-web` locally in a virtual environment as described in *OMERO.web installation for developers*, you can simply install the `minimal-webapp` example via pip:

```
$ cd omero-web-apps-examples/minimal-webapp
$ pip install -e .
```

This installs the source code directly, allowing you to work on the installed code.

You also need to add your app to the `omero.web.apps` setting:

Note: Here we use single quotes around double quotes.

```
$ omero config append omero.web.apps '"minimal_webapp"'
```

Now you can restart your `omero-web` server and go to http://localhost:4080/minimal_webapp/ in your browser. You should be redirected to the login screen and then back to the `minimal_webapp` page which will display your Name and list your Projects.

Run your app with OMERO.web in a Docker container

The following walk-through uses `omero-web-docker` to run the app. Here we add `minimal_webapp` to the installed apps and map the app directory to the `site-packages` directory in the Docker instance so that python can import our `minimal_webapp` module.

```
# You need to be in the project directory for this to work.
# cd omero-web-apps-examples/

# The OMERO server we want to connect to.
$ host=demo.openmicroscopy.org

# Path to the python module for the app.
$ appdir=$(pwd)/minimal-webapp/minimal_webapp
```

(continues on next page)

(continued from previous page)

```
# Location within Docker instance we want to link the app, so it can be imported.
$ docker_appdir=/opt/omero/web/venv3/lib/python3.6/site-packages/minimal_webapp

# This example config file installs "minimal_webapp". See the file for more details.
$ config=$(pwd)/config.omero

# Location within Docker instance we want to mount the config.
$ docker_config=/opt/omero/web/config/config.omero

# Run docker container.
$ docker run -it --rm -e OMEROHOST=$host -p 4080:4080 -v $appdir:$docker_appdir -v
↪$config:$docker_config openmicroscopy/omero-web-standalone
```

This will run Docker in the foreground, showing the output in your terminal and allowing you to kill the container with Ctrl-C. You should see the following lines in the output, indicating that OMERO.web is starting and the static files from your app are being included.

```
...
Copying '/opt/omero/web/venv3/lib/python3.6/site-packages/minimal_webapp/static/minimal_
↪webapp/app.css'
Copying '/opt/omero/web/venv3/lib/python3.6/site-packages/minimal_webapp/static/minimal_
↪webapp/app.js'
...
Starting OMERO.web...
```

Now go to http://localhost:4080/minimal_webapp/ in your browser. You should be redirected to the login screen and then back to the `minimal_webapp` page which will display your Name and list your Projects.

Exploring the app

The `minimal_webapp` code is well-documented to explain how the app is working. Briefly, the app supports a single URL defined in `minimal_webapp/urls.py` which maps to the `index` function within `minimal_webapp/views.py`. This uses the connection to OMERO, `conn` to load the current user's name and passes this to the `index.html` template to render the page.

This page also includes the static `app.js` and `app.css` files. JavaScript is used to load Projects from the *JSON API* and display them on the page.

Create an app from the template example

If you want to create your own app, you can use the example as a template.

Go to the template repository [omero-web-apps-examples](#). Click 'Use this template' as [described here](#) and choose a name for your new repo, for example `my_app`.

Go to the directory where you want your app to live and clone it. Then run as above with Docker or locally-installed OMERO.web, making sure that your app can be imported as before.

```
$ git clone https://github.com/<username>/my_app
$ cd my_app

# Then run as above...
```

App settings

You can add settings to your app that allow configuration via the command line in the same way as for the base OMERO.web. The list of CUSTOM_SETTINGS_MAPPINGS in `settings.py` is a good source for examples of the different data types and parsers you can use.

For example, if you want to create a user-defined setting `appname.foo`, that contains a dictionary of key-value pairs, you can add to CUSTOM_SETTINGS_MAPPINGS in `appname/settings.py`:

```
import json
CUSTOM_SETTINGS_MAPPINGS = {
    "omero.web.appname.foo": ["FOO", '{"key": "val"}', json.loads]
}
```

From somewhere else in your app, you can then access the settings:

```
from appname import settings
print(settings.FOO)
```

Users can then configure this on the command line as follows:

```
$ omero config set omero.web.appname.foo '{"userkey": "userval"}'
```

Linking from Webclient

If you want to add links to your app from the webclient, a number of options are described on [Linking from Webclient](#).

Releasing your app

The [Release an app](#) page has some useful steps to take when you are preparing to release your app.

3.5.4 Release an app

When you are ready to share your app with others, you can improve the install process by making your app installable via `pip`. You may also wish to configure the app label to make the app URL more user-friendly.

Make your app installable from PyPI

This is not required but it is recommended to make your app installable from PyPI. If you opt to do so, a few files need to be added:

- `setup.py` - a set-up file used to configure various aspects of the app and also used as a command line interface for packaging the app
- `setup.cfg` - a configuration file that contains option defaults for `setup.py` commands
- `MANIFEST.in` - a file needed in certain cases to package files not automatically included

See [Packaging and Distributing Projects](#) for more details.

Configuring your app name and label

We support the option of configuring your OMERO.web app with a name and label. See [Configuring Applications](#).

This allows the URL to an app to be different from its name. For example, OMERO.figure app is named `omero_figure` but the url is simply `/figure/` as configured by `__init__.py` and `apps.py`.

3.5.5 Linking from Webclient

If you want users to be able to access your app or other resources from the webclient there are a number of ways you can add links to the webclient UI.

OMERO.web top links

You can configure `omero.web.ui.top_links` to add links to the list of links at the top of the webclient main pages.

- **Name your URL in `urls.py`** (optional). Preferably we use URL names to refer to URLs. For example, the homepage of your app might be named like this in `organization-appname/urls.py`.

```
url(r'^$', views.index, name='figure_index'),
```

You can then refer to the link defined above using this name, or you can simply use a full URL for external links.

- **Update configuration** Use the OMERO command line interface to append the link to the `top_links` list.

Links use the format `["Label", "URL_name"]` or you can follow this example:

```
$ omero config append omero.web.ui.top_links '["Figure", "figure_index"]'
```

From OMERO 5.1, you can add additional attributes to links using the format `['Link Text', 'link', attrs]`. This can be used to add tool-tips and to open the link in a new “target” tab. For example:

```
$ omero config append omero.web.ui.top_links '["Homepage", "http://myhome.com", {
  ↪ "title": "Homepage", "target": "_blank"}]'
```

Custom image viewer

If you have created your own image viewer and would like to have it replace the existing image viewer in the webclient, this can be configured using `omero.web.viewer.view`.

You will need your `views.py` method to take an Image ID with a parameter named `iid`. For example, see `channel_overlay_viewer` from [omero-webtest](#) app:

```
@login_required()
def channel_overlay_viewer(request, iid, conn=None, **kwargs):
```

You can then configure the webclient to use this viewer by providing the full path name to this view method. For example, if you have `webtest` installed you can use the `channel_overlay_viewer`:

```
$ omero config set omero.web.viewer.view webtest.views.channel_overlay_viewer
```

This will now direct the image viewer url at `webclient/img_detail/<iid>/` to this viewer. However, the existing viewer will still be available under `webgateway/webgateway/img_detail/<iid>/`.

If you want to use a different viewer for different images, you can conditionally redirect to the webgateway viewer or elsewhere. For example:

```
if image.getSizeC() == 1:
    return HttpResponseRedirect(reverse("webgateway.views.full_viewer", args=(iid,)))
```

Open with

The ‘Open with’ configuration allows users to ‘open’ data from OMERO in another web page using `omero.web.open_with`.

For example:

- Open images in a custom viewer
- Open images in a new figure with `OMERO.figure`
- Link to external resources, e.g. open Dataset named ‘000397’ with url <https://www.ncbi.nlm.nih.gov/protein/000397>

Currently, ‘Open With’ options are shown in the context menu of the left-panel tree and are therefore only available for objects shown in the tree.

Label, name and supported objects

In the simplest case the minimum needed to add an Open with option is a unique identifier for your extension, a url name, and a list of the types of objects that are supported by your app. For example:

```
$ omero config append omero.web.open_with '["xyz_viewer", "url_name", {"supported_objects": ["image"]}']
```

This will create a menu option named `xyz_viewer` that is only enabled when a single “image” is selected.

The unique ID string, `xyz_viewer` can be used to identify your plugin if you add extra scripts, as shown below. If you want your Open with option to appear under a different menu label, see “UI Label” section below.

We use `reverse(url_name)` to resolve a url from the `url_name`. If the `url_name` is not recognised (for external urls) the `url_name` will be used directly.

When the `xyz_viewer` option is clicked, a new window will be opened with the selected object(s) added to the url as a query string

```
url?image=:imageId
```

Supported objects can be configured to support multiple images or other data types. Data types are `project`, `dataset`, `image`, `screen`, `plate`, `acquisition` or use plurals to indicate that multiple objects are supported. For example, `images` will enable the ‘Open with’ option when 1 or more images are selected. In the following example, we support a single dataset or multiple images.

```
"supported_objects": ["dataset", "images"]
```

Further parameters can be specified in the options object, as described below.

Open in new tab

If you wish to open in a new browser tab instead of a popup window, you can add a `target` attribute to the options:

```
$ omero config append omero.web.open_with '["xyz_viewer", "url_name", {"supported_objects
→": ["image"], "target": "_blank"}]'
```

UI Label

If a “label” is specified in the options object, this will be used as the display label in the webclient context menu instead of using the ID.

```
$ omero config append omero.web.open_with '["xyz_viewer", "url_name"], {"supported_
→objects": ["image"], "label": "X-Y-Z viewer"}]'
```

JavaScript handlers

For more control over the enabled status of your plugin or to configure how urls are created from selected objects, you can write JavaScript functions that handle these steps. These functions use the label specified above as an ID for your `Open with` option. In this example it is `xyz_viewer`. Add one or both of these function calls to a script, for example `openwith.js`

```
// Here we set an 'enabled' handler that is passed a list of selected
// objects and should return ``true`` if the 'Open with' option should
// be enabled.
// The ``supported_objects`` parameter will not be needed.
// First argument is the label that we used above to identify the option
OME.setOpenWithEnabledHandler("xyz_viewer", function(selected){
    // selected is a list of objects containing id, name, type

    // Only support single objects
    if (selected.length !== 1) return false;

    // Only support image with name ending in .svs
    var obj = selected[0];
    return (obj.type === 'image' && obj.name.endsWith('.svs'))
});

// Here we configure a url provider. This function will be passed the selected
// objects and the base url that was specified in the 'Open with' configuration above.
OME.setOpenWithUrlProvider("xyz_viewer", function(selected, url) {

    // Build a url using id from selected objects
    url += selected[0].id + "/";
    return url;
});
```

Note that instead of returning a static URL, you can also return a function, which will be called when the menu option is selected:


```
// Here we configure a url provider that returns a function instead of
// a static URL. As an example, this shows an alert instead of opening
// a new web page.
OME.setOpenWithUrlProvider("xyz_viewer", function(selected, url) {
  return function () {
    window.alert("The first selected ID is " + selected[0].id);
  };
});
```

Save the script to a static location, either within an OMERO.web app's static directory or make it available at another url. Then specify this location using the `script_url` option.

Note: Once you have added a script and updated the config, you will need to restart OMERO.web as normal. This will syncmedia to copy the script to the static files location.

```
# Script is saved at myviewer/static/myviewer/openwith.js
$ omero config append omero.web.open_with '["xyz_viewer", "url_name"], {"script_url":
↪ "myviewer/openwith.js"}]'

# 'Open with' option loads a script from the specified url.
# The script will open any object with url https://www.ncbi.nlm.nih.gov/protein/:name
# and is enabled when the :name of the object is a number (all digits)
$ omero config append omero.web.open_with '["GenBank Protein", "https://www.ncbi.nlm.nih.
↪ gov/protein/", {"script_url": "https://will-moore.github.io/presentations/2016/
↪ OpenWith-Filtering-June-2016/openwith.js"}]'
```

OMERO.web plugins

If you want to display content from your app within the webclient UI, please see [Webclient Plugins](#).

3.5.6 Webclient Plugins

The webclient UI can be configured to include content from other web apps. This allows you to extend the webclient UI with your own functionality. This is used by the webtagging apps [autotag](#) and [tagsearch](#), and there are also some examples in the [omero-webtest](#) repository.

Currently you can add content in the following locations:

- **Center Panel** Adding a panel to the center of the webclient will display a drop-down menu to the top right of the center panel, allowing users to choose your plugin.
- **Right Panel** You can add additional tabs to the right panel. These will be available in the main webclient page as well as history and search result pages.

Overview

To begin with, you need to prepare your plugin pages in your own app, with their own URLs, views and templates. Then you can display these pages within the webclient UI, using the plugin framework.

Note: When you embed your pages in the webclient, there is potential for conflicts between JavaScript and CSS in the host page and your own code. Care must be taken not to overwrite global JavaScript functions (such as jQuery or the 'OME' namespace) or to add CSS codes that may affect host elements.

The webclient plugins work by adding some custom JavaScript snippets into the main pages of the webclient and adding HTML elements to specified locations in the webclient. These snippets of JavaScript can be used to load content into these HTML elements. Usually you will want to do this dynamically, to display data based on the currently selected objects (although this is optional). Helpers can be used to respond to changes in the selected objects and the selected tab, so you can load or refresh your plugin only when necessary.

App URLs

To display content based on currently selected data, such as Projects, Datasets and Images, your app pages will need to have these defined in their URLs. For example:

```
# Webtagging: Tag images within the selected dataset
url(r'^auto_tag/dataset/(?P<datasetId>[0-9]+)/$', views.auto_tag),

# Webtest: Show a panel of ROI thumbnails for an image
url(r'^image_rois/(?P<imageId>[0-9]+)/', views.image_rois, name='webtest_image_rois'),
```

These URLs should simply display the content that you want to show in the webclient. When these pages load in the webclient, they will have all the webclient CSS and JavaScript (such as jQuery) available so you do not need to include these in your page.

Configuring the plugin

Choose an element ID

You will need to specify an ID for the `<div>` element that is added to the webclient, so that you can refer to this element in the JavaScript. For example, `image_roi_tab` or `auto_tag_panel`.

Create a JavaScript file

This will contain the JavaScript snippet that is injected into the main webclient page `<head>` when the page is generated. This is added using Django's templates, so it should be placed within your app's `/templates/<organization_appname>/` directory and named `.html`, e.g. `/templates/<organization_appname>/webclient_plugins/right_plugin_rois.html`. All the JavaScript should be within `<script>` and `</script>` tags. Your plugin initialization should happen after the page has loaded, so you use the jQuery on-ready function.

You use custom jQuery functions, called `omeroweb_right_plugin` or `omeroweb_center_plugin`, to initialize the webclient plugin. These will handle all the selection change events. You simply need to specify how the panel is loaded, based on the selected object(s) and what objects are supported. The plugin will be disabled when non-supported objects are selected.

Below is a simple example of their usage. More detailed documentation available in the [plugin options section](#) below.

Center Panel Plugin

```
<script>
$(function() {

    // Initialise the center panel plugin, on our specified element
    $("#auto_tag_panel").omeroweb_center_plugin({

        // To support single item selection, we can specify the types like this.
        // Tab will only be enabled when a single dataset is selected
        supported_obj_types: ['dataset'],

        load_plugin_content: function(selected, dtype, oid){

            // since we currently limit our dtype to 'dataset', oid will be dataset ID
            // Use the 'index' of your app as base for your URL
            var auto_tag_url = '{% url 'webtagging_index' %}auto_tag/dataset/'+oid+'/';
            $(this).load(auto_tag_url);

        }

    });
});
</script>
```

Right Tab Plugin

```
<script>
$(function() {

    // Initialise the right tab plugin, on our specified tab element
    $("#image_roi_tab").omeroweb_right_plugin({

        // Tab will only be enabled when a single image is selected
        supported_obj_types: ['image'],

        // This will get called when tab is displayed or selected objects change
        load_plugin_content: function(selected, obj_dtype, obj_id) {

            // since we only support single images, the obj_id will be an image ID
            // Generate url based on a template-generated url
            var url = '{% url 'webtest_index' %}image_rois/' + obj_id + '/';

            // Simply load the tab
            $(this).load(url);

        },

    });

});
</script>
```

Plugin installation

Now you need to add your plugin to the appropriate plugin list, stating the displayed name of the plugin, the path/to/js_snippet.html and the ID of the plugin element. Plugin lists are:

- omero.web.ui.center_plugins
- omero.web.ui.right_plugins

Use the OMERO command line interface to add the plugin to the appropriate list.

```
$ omero config append omero.web.ui.center_plugins
  ["Auto Tag", "omero_autotag/auto_tag_init.js.html", "auto_tag_panel"]'
```

The right_plugins list includes the *Acquisition* tab and *Preview* tab by default. If you want to append the OMERO.webtest ROI plugin or your own plugin to the list, you can simply do:

```
$ omero config append omero.web.ui.right_plugins
  ["ROIs", "omero_webtest/webclient_plugins/right_plugin.rois.js.html", "image_roi_tab
↪"]'
```

If you want to replace existing plugins and display only your own single plugin, you can simply do:

```
$ omero config set omero.web.ui.right_plugins
  [[["ROIs", "omero_webtest/webclient_plugins/right_plugin.rois.js.html", "image_roi_
↪tab"]]]'
```

Restart Web

Stop and restart your web server, then refresh the webclient UI. You should see your plugin appear in the webclient UI in the specified location. You should only be able to select the plugin from the drop-down menu or tab **if** the supported data type is selected, e.g. 'image'. When you select your plugin, the load content method you specified above will be called and you should see your plugin loaded.

Refreshing content

If you now edit the views.py or HTML template for your plugin and want to refresh the plugin within the webclient, all you need to do is to select a different object (e.g. dataset, image etc.). If you select an object that is not supported by your plugin, then nothing will be displayed, and for the right-tab plugin, the tab selection will change to the first tab.

Plugin options

- **supported_obj_types:** If your plugin displays data from single objects, such as a single Image or Dataset, you can specify that here, using a list of types:

```
supported_obj_types: ['dataset', 'image'],
```

This will ensure that the plugin is only enabled when a single Dataset or Image is selected. To support multiple objects, see 'tab_enabled'.

- **plugin_enabled:** This function allows you to specify whether a plugin is enabled or not when specified objects are selected. It is only used if you have NOT defined 'supported_obj_types'. The function is passed a single argument:

- selected: This is a list of the selected objects e.g. `[{'id': 'image-123'}, {'id': 'image-456'}]`

The function should return true if the plugin should be enabled. For example, if you want the center plugin to support multiple images, or a single dataset:

```
plugin_enabled: function(selected){
  if (selected.length == 0) return false;
  var dtype = selected[0]['id'].split('-')[0];
  if (selected.length > 1) {
    return (dtype == "image");
  } else {
    return ($.inArray(dtype, ["image", "dataset"]) > -1);
  }
}
```

- **load_plugin_content / load_tab_content:** This function will be called when the plugin/tab content needs to be refreshed, either because the plugin is displayed for the first time, or because the selected object changes. The function will be passed 3 arguments:

- selected: This is a list of the selected objects e.g. `[{'id': 'image-123'}, {'id': 'image-456'}]`
- obj_dtype: This is the data-type of the first selected object, e.g. 'image'
- obj_id: This is the ID of the first selected object, e.g. 123

3.5.7 Editing OMERO.web

If you need to make changes to OMERO.web itself, then you can perform a developer install of `omero-web`. You need to be within a virtual environment with `omero-py` installed as described at [OMERO Python language bindings](#). Then:

```
$ git clone https://github.com/ome/omero-web.git
$ cd omero-web
$ pip install -e .
```

This will allow you to edit and run the source code without a build step.

You can then run OMERO.web as described at [OMERO.web installation for developers](#). You may need to restart the web server after saving changes, particularly for python files, before refreshing the browser.

3.5.8 WebGateway

WebGateway is a Django app within the [OMERO.web framework](#). It provides a web API for rendering images and accessing data on the OMERO server via URLs.

Note: The OMERO.web client also supports URLs linking to specified data in OMERO. See the [OMERO.figure User guide](#) for more details.

Web services

This list of URLs below may be incomplete or out of date. For a complete list of URLs, see the [latest API](#) and try the URLs out for yourself!

The HTTP request will need to include login details for creating or using a current server connection. This will be true for any request made after logging in to the server, e.g. login using the webclient login page then go to `webgateway/...` or if you have logged in to a server at `http://ome.example.com/webclient` then go to, for example, `http://ome.example.com/webgateway/render_image/<imageid>/<z>/<t>/`

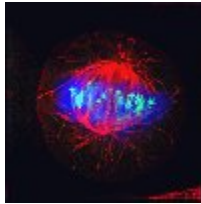


Fig. 8: Rendered thumbnail

URLs from within OMERO.web

Images rendered within OMERO.web templates should use Django's `{% url %}` tag to generate URLs for webgateway, passing in the ID of the image. This is shown for each of the URLs below:

Image viewer

- Provides a full image viewer, with controls for scrolling Z and T, editing rendering settings etc.

URL: `https://your_host/webgateway/img_detail/<imageid>/`

Template tag: `{% url 'webgateway_full_viewer' image_id %}`

Images

- Returns a jpeg of the specified plane with current rendering settings

URL: `https://your_host/webgateway/render_image/<imageid>/<z>/<t>/`

Template tag: `{% url 'webgateway_render_image' image_id theZ theT %}`

Omitting Z and T will use the default values:

URL: `https://your_host/webgateway/render_image/<imageid>/`

Template tag: `{% url 'webgateway_render_image' image_id %}`

Partially deprecated: The desired rendering settings for individual channels can be specified in the query string. It is recommended that either none or all of the channels are specified explicitly. Specifying partial channel lists is deprecated.

- Makes a jpeg laying out each active channel in a separate panel

URL: `https://your_host/webgateway/render_split_channel/<imageId>/<z>/<t>/`

Template tag: `{% url 'webgateway_render_split_channel' image_id theZ theT %}`

- Plots the intensity of a row of pixels in an image. `w` is line width

URL: `https://your_host/webgateway/render_row_plot/<imageId>/<z>/<t>/<y>/<w>`

Template tag: `{% url 'webgateway_render_row_plot' image_id theZ theT yPos width %}`

- Plots the intensity of a column of pixels in an image.

URL: `https://your_host/webgateway/render_col_plot/<imageId>/<z>/<t>/<x>/<w>/`

Template tag: `{% url 'webgateway_render_col_plot' image_id theZ theT xPos width %}`

- Returns a jpeg of a thumbnail for an image. `w` and `h` are optional (default is 64). Specify just one to retain aspect ratio. It is also possible to specify `Z` and `T` indices in the query string.

URL: `https://your_host/webgateway/render_thumbnail/<imageId>/?z=10`

Template tag: `{% url 'webgateway_render_thumbnail' image_id %}?z=10 # default_`
`→size, z=10`

URL: `https://your_host/webgateway/render_thumbnail/<imageId>/<w>/<h>`

Template tag: `{% url 'webgateway_render_thumbnail' image_id 100 %} # size 100`

Rendering settings

If no rendering settings are specified (as above), then the current rendering settings will be used. To apply different settings to images returned by the `render_image` and `render_split_channels` URLs, parameters can be specified in the request. N.B. These settings are only applied to the rendered image and will not be saved unless specified.

Individual parameters are:

- **Channels on/off** e.g. for an image with 4 channels, to turn on all channels except 2:

`?c=1,-2,3,4`

You can simply specify the active channels.

`?c=3 # only Channel 3 is active`

`?c=3,4 # Channels 3 and 4 are active`

- **Channel color** e.g. to set the colors for channels 1 to red and 2 to green and 3 to blue:

`?c=1|$FF0000,2|$00FF00,3|$0000FF`

- **Rendering levels** e.g. to set the cut-in and cut-out values for an image with 3 Channels.

`?c=1|400:505,2|463:2409,3|620:3879`

`?c=-1|400:505,2|463:2409,3|620:3879 # First channel inactive "-1"`

`?c=2|463:2409,3|620:3879 # Inactive channels can be omitted`

- **Z-projection:** Maximum intensity, Mean intensity or None (normal). By default we use all z-sections, but a range can be specified.

```
?p=intmax
?p=intmax|0:10      # Use z-sections 0-10 inclusive
?p=intmean
?p=normal
```

- **Rendering ‘Mode’:** greyscale or color.

```
?m=g      # greyscale (only the first active channel will be shown in grey)
?m=c      # color
```

- **Codomain maps:** OMERO’s rendering engine supports mapping from input -> output pixel intensity via application of “codomain maps”. Currently only the ‘reverse’ intensity map is supported, but the use of JSON encoding for the maps query parameter is designed to support more maps in future. In the case of the ‘reverse’ map, we only need to specify whether it is enabled for each channel. For an image with 2 channels, to enable reverse map for the first channel, we can use this query string:

```
?maps=[{"reverse":{"enabled":true}},{"reverse":{"enabled":false}}]
```

- Parameters can be combined, e.g.

```
https://your_host/webgateway/render_image/2602/10/0/?c=1|100:505$0000FF,2|463:2409
↪$00FF00,3|620:3879$FF0000,-4|447:4136$FF0000&p=normal
```

JSON methods

- List of projects: `webgateway/proj/list/`

```
[{"description": "", "id": 269, "name": "Aurora"},
{"description": "", "id": 269, "name": "Drugs"} ]
```

- Project info: `webgateway/proj/<projectId>/detail/`

```
{"description": "", "type": "Project", "id": 269, "name": "CenpA"}
```

- List of Datasets in a Project: `webgateway/proj/<projectId>/children/`

```
[{"child\_count": 9, "description": "", "type": "Dataset", "id": 270,
  "name": "Control"}, ]
```

- Dataset, same as for Project: `webgateway/dataset/<datasetId>/detail/`

- Details of Images in the dataset: `webgateway/dataset/<datasetId>/children/`

- Lots of metadata for the image. See below: `webgateway/imgData/<imageId>/`

- **Histogram** of pixel intensity data for an image plane. Channel index is zero-based. By default the Z and T index are 0 and the number of histogram bins is 256, but these can be specified in the query string. The range of the histogram will be the pixel intensity range for that channel of the image (see “window”: “min” and “max” in `imgData` below)

```
URL: webgateway/histogram_json/<imageId>/channel/<index>/?theT=0&theZ=0&bins=20
```



```
{
  "data": [ 24354, 93878, 87555, 45323, 27365, 14690, 9346, 2053, 60, 7, 19, 14, 15, ↵
    ↵9, 5, 5, 3, 0, 2, 1]
}
```

Saving etc.

- `webgateway/saveImgRDef/<imageId>/`
- `webgateway/compatImgRDef/<imageId>/`
- `webgateway/copyImgRDef/`

ImgData

The following is sample JSON data generated by `/webgateway/imgData/<imageId>/`

```
{
  "split_channel": {
    "c": {"width": 1448, "gridy": 2, "border": 2, "gridx": 3, "height": 966},
    "g": {"width": 966, "gridy": 2, "border": 2, "gridx": 2, "height": 966}
  },
  "rdefs": {"defaultT": 0, "model": "color",
    "projection": "normal", "defaultZ": 15},
  "pixel_range": [-32768, 32767],
  "channels": [
    {"color": "0000FF", "active": true,
      "window": {"max": 449.0, "end": 314, "start": 70, "min": 51.0},
      "emissionWave": "DAPI",
      "label": "DAPI"},
    {"color": "00FF00", "active": true,
      "window": {"max": 7226.0, "end": 1564, "start": 396, "min": 37.0},
      "emissionWave": "FITC",
      "label": "FITC"}
  ],
  "meta": {
    "projectDescription": "",
    "datasetName": "survivin",
    "projectId": 2,
    "imageDescription": "",
    "imageTimestamp": 1277977808.0,
    "imageId": 12,
    "imageAuthor": "Will Moore",
    "imageName": "CSFV-siRNAi02_R3D_D3D.dv",
    "datasetDescription": "",
    "projectName": "siRNAi",
    "datasetId": 3
  },
  "id": 12,
  "pixel_size": {"y": 0.0663, "x": 0.0663, "z": 0.2},
  "size": {
    "width": 480,
    "c": 4,

```

(continues on next page)

(continued from previous page)

```

    "z": 31,
    "t": 1,
    "height": 480
  },
  "tiles": false
}

```

For large tiled images, the following data is also included:

```

{
  "tiles": true,
  "tile_size": {
    width: 256,
    height: 256
  },
  "levels": 5,
  "zoomLevelScaling": {
    0: 1,
    1: 0.25,
    2: 0.0625,
    3: 0.0312,
    4: 0.0150
  },
}

```

3.5.9 Embedding an OMERO.web viewport in a web page

Note: These example are intended to be used with images that have been added to the PUBLIC group with a Public member in OMERO, making them publicly available. To see how to configure public URL filters, see the [Publishing data using OMERO.web](#) section.

OMERO.web viewer in iframe

Insert the following:

```

<div id="omeroviewport"><iframe width="850" height="600" src="http://localhost:8000/
↪webclient/img_detail/IMAGE_ID/" id="omeroviewport" name="omeroviewport"></iframe></div>

```

Launching OMERO.web viewer

Use the following code to reference the scripts.

```

<script type="text/javascript">

  function openPopup(url) {
    owindow = window.open(url, 'anew', config='height=600,width=850,left=50,top=50,
↪toolbar=no,menubar=no,scrollbars=yes,resizable=yes,location=no,directories=no,status=no

```

(continues on next page)

(continued from previous page)

```

    ↪');
    if(!owindow.closed) owindow.focus();
    return false;
  }
</script>

```

Then in <BODY> insert the following:

```

<a href="#" onclick="return openPopup('http://localhost:8000/webclient/img_detail/IMAGE_
↪ID/')">Open viewer</a>

```

Customizing the content of the embedded OMERO.web viewport

This section demonstrates how to embed an OMERO.web image viewer in any HTML page, allowing use of resources directly from an OMERO server.

```
$ omero config set omero.web.public.url_filter '^/webgateway'
```

Provided the image corresponding to IMAGE_ID is in the PUBLIC group, it can be accessed via the link: *http://your_host/webgateway/img_detail/IMAGE_ID/*. Please remember that public images must be in a public group where a public user can access them. The *Publishing data using OMERO.web* documentation section can help you to set this up.

Use the following code to load stylesheets and scripts.

```

<link rel="stylesheet" type="text/css" href="http://your_host/static/omeroweb.viewer.min.
↪css">
<script type="text/javascript" src="http://your_host/static/omeroweb.viewer.min.js"></
↪script>

```

Then create the small JavaScript with associated stylesheet which allows you to view particular image defined by IMAGE_ID.

```

<style type="text/css">
  .viewport {
    height: 500px;
    width: 800px;
    padding: 10px;
  }
</style>

<script type="text/javascript">
  $(document).ready(function () {

    /* Prepare the viewport */
    viewport = $.WeblitzViewport($("#viewport"), "http://your_host/webgateway/", {
      'mediaroot': "http://your_host/static/"
    });

    /* Load the selected image into the viewport */
    viewport.load(IMAGE_ID);
  });

```

(continues on next page)

(continued from previous page)

```
});
</script>
```

Then in <BODY> insert the following:

```
<div id="viewport" class="viewport"></div>
```

The viewport can be made more interactive by adding buttons or links to allow display of scalebars, ROIs, zooming and selection of channels. Full examples of how to embed microscopy or Whole Slide Image are available in the [OMERO.webtest GitHub repository](#).

3.5.10 Writing OMERO.web views

This page contains info on how to write your own views.py code, including documentation on the `webclient/views.py` and `webgateway/views.py` code. Although we aim to provide some useful notes and examples here, you will find the best source of examples is [the code itself](#).

@Decorators

Decorators in Python are functions that ‘wrap’ other functions to provide additional functionality. They are added above a method using the @ notation. We use them in the OMERO.web framework to handle common tasks such as login (getting connection to OMERO server) etc.

@login_required()

The `login_required` decorator uses parameters in the ‘request’ object to retrieve an existing connection to OMERO. In the case where the user is not logged in, they are redirected to a login page. Upon login, they will be redirected back to the page that they originally tried to view. The method that is wrapped by this decorator will be passed a ‘conn’ Blitz Gateway connection to OMERO.

Note: `login_required` is a class-based decorator with several methods that can be overwritten to customize its functionality (see below). This means that the decorator **MUST** be instantiated when used with the @ notation, i.e.

```
@login_required()    NOT    @login_required    # this will give you strange error messages
```

A simple example of `@login_required()` usage (in `omerowebtest/views.py`). Note the Blitz Gateway connection “conn” retrieved by `@login_required()` is passed to the function via the optional parameter `conn=None`.

```
from omeroweb.decorators import login_required

@login_required()
def dataset(request, datasetId, conn=None, **kwargs):
    ds = conn.getObject("Dataset", datasetId)
    return render(request, 'webtest/dataset.html', {'dataset': ds})
```

or

```

from omeroweb.decorators import login_required, render_response

@login_required()
@render_response()
def dataset(request, datasetId, conn=None, **kwargs):
    ds = conn.getObject("Dataset", datasetId)
    context['template'] = 'webtest/dataset.html'
    context['dataset'] = ds
    return context

```

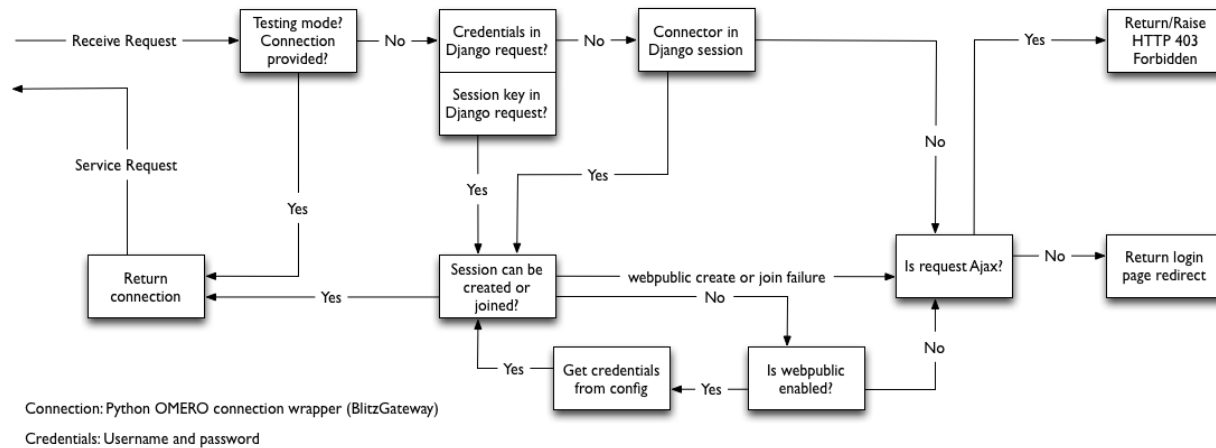


Fig. 9: Logic flow for retrieving Blitz Gateway connection from HTTP request.

login_required logic

The `login_required` decorator has some complex connection handling code, to retrieve or create connections to OMERO. Although it is not necessary to study the code itself, you may find it useful to understand the logic that is used (see Flow Diagram). As mentioned above, we start with a HTTP request (top left) and either a connection is returned (bottom left) OR we are redirected to login page (right).

Note: Options to configure a “public user” are described in the *Publishing data using OMERO.web* documentation.

Extending login_required

The base `login_required` class can be found in `omeroweb/decorators.py`. It has a number of methods that can be overwritten to customize or extend its functionality. Again, it is best to look at an example of this. See `webclient/decorators.py` to see how the base `omeroweb.decorators.login_required` has been extended to configure the conn connection upon login, handle login failure differently etc.

render_response

This decorator handles the rendering of view methods to `HttpResponse`. It expects that wrapped view methods return a dict. This allows:

- the template to be specified in the method arguments OR within the view method itself
- the dict to be returned as json if required
- the request is passed to the template context, as required by some tags etc
- a hook is provided for adding additional data to the context, from the [Blitz Gateway documentation](#) or from the request.

Note: Using `@render\_response` guarantees using special `RequestContext` class which always uses `django.template.context_processors.csrf` (no matter what template context processors are configured in the `TEMPLATES` setting). For more details see [CSRF](#).

Extending render_response

The base `render_response` class can be found in `omeroweb/decorators.py`. It has a number of methods that can be overwritten to customize or extend its functionality. Again, it is best to look at an example of this. See `webclient/decorators.py` to see how the base `omeroweb.decorators.render_response` has been extended to configure `HttpResponse` and its subclasses.

Style guides

Tips on good practice in `views.py` methods and their corresponding URLs.

- Include any required arguments in the function parameter list. Although many `views.py` methods use the **kwargs parameter to accept additional arguments, it is best not to use this for arguments that are absolutely required by the method.**
- Specify default parameters where possible. This makes it easier to reuse the method in other ways.
- Use keyword arguments in URL regular expressions. This makes them less brittle to changes in parameter ordering in the views.
- Similarly, use keyword arguments for URLs in templates

```
{% url 'url_name' object_id=obj.id %}
```

and reverse function:

```
>>> from django.urls import reverse
>>> reverse('url_name', kwargs={'object_id': 1})
```

OMERO.web error handling

Django comes with some nice error handling functionality. We have customized this and also provided some client-side error handling in JavaScript to deal with errors in AJAX requests. This JavaScript can be included in all pages that require this functionality. Errors are handled as follows:

- **404** - simply displays a 404 message to the user
- **403** - this is ‘permission denied’ which probably means the user needs to login to the server (e.g. session may have timed out). The page is refreshed which will redirect the user to login page.
- **500** - server error. We display a feedback form for the user to submit details of the error to our QA system - POSTs to “qa.openmicroscopy.org.uk:80”. This URL is configurable in `settings.py`.

In general, you should not have to write your own error handling code in `views.py` or client side. The default behavior is as follows:

With Debug: True (during development)

Django will return an HTML page describing the error, with various parameters, stack trace etc. If the request was AJAX, and you have our JavaScript code on your page then the error will be handled as described (see above). NB: with Debug True, 500 errors will be returned as HTML pages by Django but these will not be rendered as HTML in our feedback form. You can use developer tools on your browser (e.g. Firebug on Firefox) to see various errors and open the request in a new tab to display the full debug info as HTML.

With Debug: False (in production)

Django will use its internal error handling to produce standard 404, 500 error pages. We have customized this behavior to display our own error pages. The 500 error page allows you to submit the error as feedback to our QA system. If the request is AJAX, we return the stack trace is displayed in a dialog which also allows the error to be submitted to QA.

Custom error handling

If you want to handle certain exceptions in particular ways you should use appropriate try/except statements.

This is only advised for trivial errors, where you can give the user a simple message, e.g. “No Objects selected, please try again”, or if the error is well understood and you can recover from the error in a reasonable way.

For ‘unexpected’ server errors, it is best to allow the exception to be handled by Django since this will provide a lot more info to the user (request details etc.) and format HTML (both with Debug True or False).

If you still want to handle the exception yourself, you can provide stack trace alongside a message for the user. If the request is AJAX, do not return HTML, since the response text will be displayed in a dialog box for the user (not rendered as HTML).

```
try:
    # something bad happens
except:
    # log the stack trace
    logger.error(traceback.format_exc())
    # message AND stack trace
    err_msg = "Something bad happened! \n \n%s" % traceback.format_exc()
    if request.is_ajax():
        return HttpResponseServerError(err_msg)
```

(continues on next page)

(continued from previous page)

```

else:
    ... # render err_msg with a custom template
    return HttpResponseRedirect(content)

```

3.5.11 Writing page templates in OMERO.web

This page documents the various base templates that are used by the webclient and describes how to extend these to create your own pages with the OMERO.web look and feel.

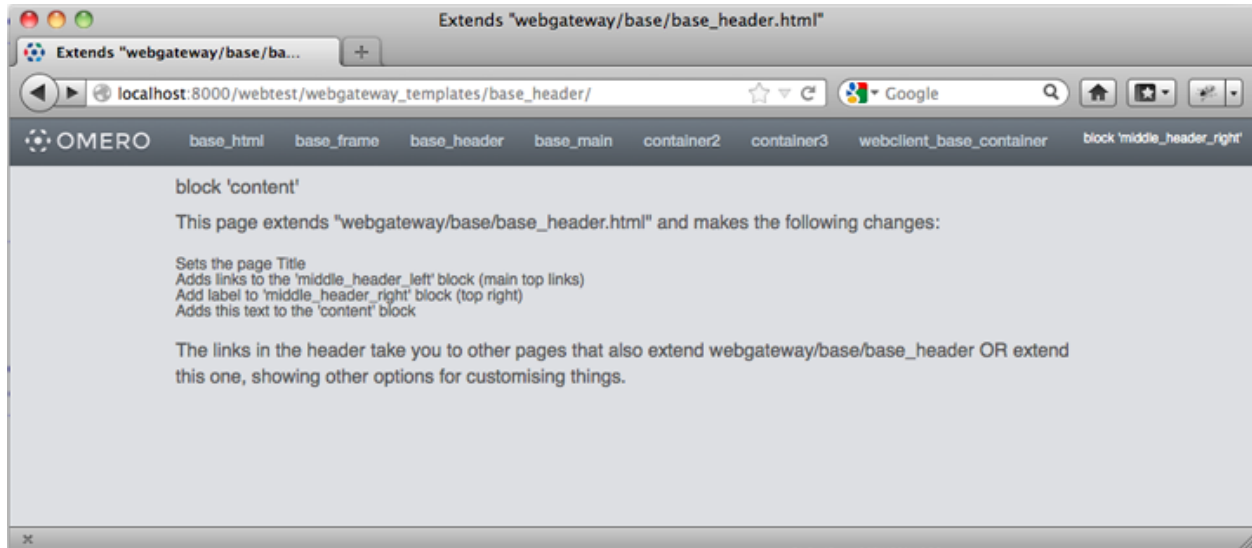


Fig. 10: The `base_header.html` template extended in OMERO.webtest with dummy content

You can use these templates in a number of ways, but there are 2 general scenarios that are detailed below:

- You want a page header to look like the webclient, but you do not need any data or connection to an OMERO server.
- You want a page that looks and behaves like it is part of the webclient application, including data from the OMERO server.

Django templates

We use Django templates for the OMERO.web pages. See docs here: [templates](#). We have designed a number of OMERO.web base templates that you can extend. The base templates live in the 'webgateway' app under `omeroweb/webgateway/templates/webgateway/base`. You can use these to make pages that do not require an OMERO login (e.g. public home page) etc.

If you want your pages to extend the webclient application, you can use templates from `omeroweb/webclient/templates/webclient/base`.

These templates are described in more detail below.

Getting Started

Within your *OMERO.web* app, create a new page template and add this line at the top:

```
{% extends "webgateway/base/base_header.html" %}
```

Now add the page content in a ‘content’ block like this:

```
{% block content %}
    Your page content goes here
{% endblock %}
```

You can now save this template and view the page. It should look something like the screen-shot above. You could add a ‘title’ block to set the page <title>

```
{% block title %}
    My OMERO.web app page
{% endblock %}
```

Additional blocks can be used to customize the page further. See below for more details.

Using Webclient templates

Webclient templates can be used in exactly the same way, for example try using this at the top of the page you created above:

```
{% extends "webclient/base/base_container.html" %}
```

However, this template will need various pieces of data to be in the page context that Django uses to render the page. You will need to use the `@login_required()` and `@render_response()` decorators on your `views.py` methods in order to retrieve this info and pass it to the template. See [Writing OMERO.web views](#) for more details.

If you have used the ‘content’ block on this page (as described above) you will see that your page content fills the whole area under the header. However, if you want to use the same 3 column layout as the webclient, you can replace your ‘content’ block with:

```
{% block left %}
    Left column content
{% endblock %}

{% block center %}
    Center content
{% endblock %}

{% block right %}
    Right column content
{% endblock %}
```

This should give you something like the screen-shot below.

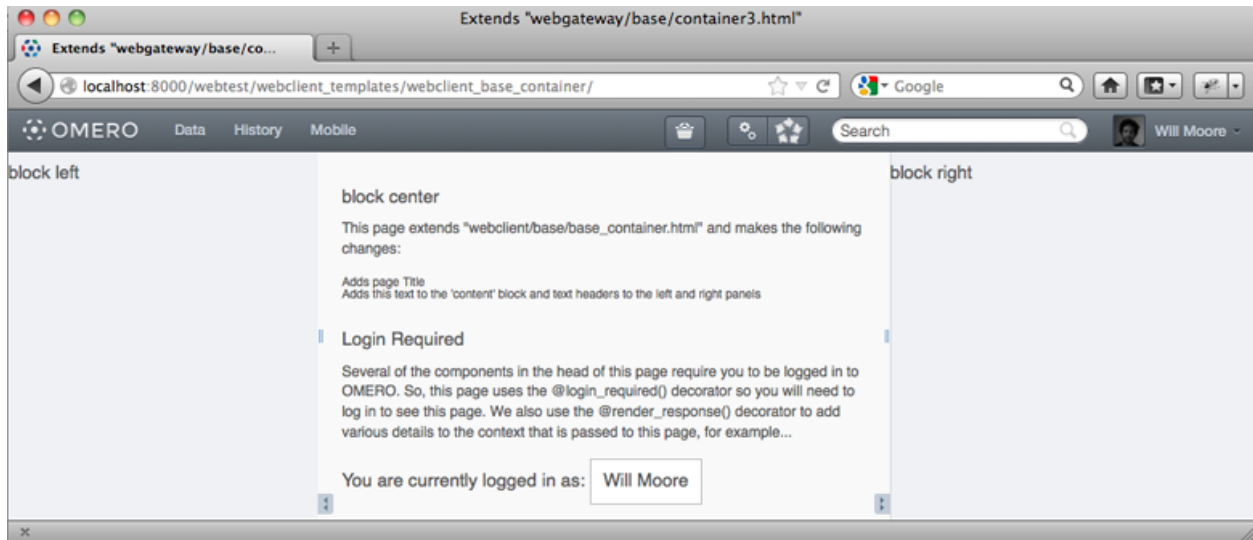


Fig. 11: The `webclient/base/base_container.html` template extended in OMERO.webtest with dummy content

Extending templates

You should aim to create a small number of your own base templates, extending the OMERO.web webgateway or webclient templates as required. If you extend all of your own pages from a small number of your own base templates, then you will find it easier to change things in future. For example, any changes in our ‘webgateway’ templates will only require you to edit your own base templates.

Here is a full list of the templates under `omeroweb/webgateway/templates/webgateway/base` with more details below:

- **base_html.html** - This provides the base `<html>` template with blocks for ‘link’ (for CSS) ‘title’ ‘script’ and ‘body’. It is extended by every other template. Usage: `{% extends "webgateway/base/base_html.html" %}`
- **base_frame.html** - This adds jQuery and jQuery-ui libraries to a blank page. Used for popup windows etc. Usage: `{% extends "webgateway/base/base_frame.html" %}`
- **base_header.html** - This also extends `base_html.html` adding all the header and footer components that are used by the webclient. See screen-shot above. More details below.
- **base_main.html** - This adds jQuery and jQuery-ui libraries to the `base_header.html` template. Used for popup windows etc. Usage: `{% extends "webgateway/base/base_main.html" %}`
- **container2.html, container3.html** - These templates extend the `base_header.html` template, adding a 2 or 3 column layout in the main body of the page. `container3.html` is used by the webclient for the `base_container` example above.

Webtest examples

You can find examples of how to extend the base templates in the [OMERO.webtest](#) repository within the `omero_webtest/templates/webtest/webgateway` directory. If you install the OMER0.webtest app, you can view the template examples live at `<your-server-name>/webtest/webgateway_templates/base_header/>`

The link is to an example that extends `base_header.html` and contains links to all the other webtest examples. These pages indicate the names of the various template “blocks” that have been used to add content to different parts of the page (also see below for block names).

Content blocks

These blocks can be used to add content to specific points in the page.

Note: It is important to consider using `{{ block.super }}` if you want to include the content from the parent template. This is critical for the “link” and “script” blocks, which are used to add `<link>` and `<script>` elements to the head of the page. If you forget to use ``` {{ block.super }} ``` then you will remove all the CSS and JavaScript links required by the parent template.

See [base_header.html](#) for full template details.

- link: used to add CSS with `<link>` blocks to the page head e.g.

```
{% block link %}
    {{ block.super }}
    <link rel="stylesheet" type="text/css"
        href="{% static "webgateway/css/ome.body.css" %}" />
{% endblock %}
```

- script - used to add JavaScript with `<script>` blocks to the page head
- title - add text here for the page `<title>`.
- head - another block for any extra head elements
- middle_header_right - add content to the right of the main header
- middle_header_left - add content to the left of the main header
- content - main page content.

container2.html, container3.html

These templates have all the same blocks as `base_header.html` since they extent it (see above). In addition, they also add the following blocks:

- left: The left column (NOT in `container2.html`)
- center: The middle column
- right: The right column

See [container3.html](#) for full template details.

3.5.12 Cross Site Request Forgery protection

CSRF is an attack which forces an end user to execute unwanted actions on a web application in which they are currently authenticated. For more details see [Cross-Site Request Forgery](#).

OMERO.web provides easy-to-use protection against Cross Site Request Forgeries, for more information see [Django documentation](#).

The first defense against CSRF attacks is to ensure that GET requests (and other ‘safe’ methods, as defined by 9.1.1 Safe Methods, HTTP 1.1, [RFC 2616](#)) are only reading data from the server.

Requests that write data to the server should only use methods such as POST, PUT and DELETE. These requests can then be protected as follows:

- By default OMERO.web has the middleware `django.middleware.csrf.CsrfViewMiddleware` added to the list of middleware classes.
- In any template that uses a POST form, use the `csrf_token` tag inside the `<form>` element if the form is for an internal URL, e.g.:

```
<form action="." method="post">{% csrf_token %}
```

Note: This should not be done for POST forms that target external URLs, since that would cause the CSRF token to be leaked, leading to a vulnerability.

- On each XMLHttpRequest set a custom X-CSRFToken header to the value of the CSRF token and pass the CSRF token in data with every AJAX POST request. You can import a jQuery-based script to do this as follows:

```
<script type="text/javascript" src="{% static "webgateway/js/ome.csrf.js" %}"></script>
```

For more details see [CSRF for ajax](#).

The Django framework also offers decorator methods that can help you protect your view methods and restrict access to views based on the request method. For more details see [Django decorators](#).

By default OMERO.web provides a built-in view that handles all unsafe incoming requests failing with **403 Forbidden** response if the CSRF token has not been included with a POST form.

3.5.13 The OMERO.web client application

The webclient Django app provides the main OMERO.web client UI.

Data is retrieved from OMERO using the *OMERO Python language bindings* and is rendered to HTML using [Django templates](#) before being sent to the browser. Additional javascript (jQuery-based) functionality in the browser is used to update the UI in response to user interactions such as browsing or modifying the data. This will often involve AJAX calls to load more data as HTML or JSON objects.

Note: The webclient should **NOT** be considered as a stable public API. URLs and methods within the app are purely designed to provide internal functionality for the webclient itself and may change in minor releases without warning.

Top level pages

There are a small number of top level HTML pages that the user will start at. These are all handled by the `load_template` view:

- `/` (homepage delegates to `/userdata`)
- `/userdata`
- `/usertags`
- `/public`
- `/history`
- `/search`

These pages contain many different jQuery scripts that run when the page loads, to setup event listeners and to load additional data.

Additional top-level pages are used in popup windows for running scripts and downloading data.

JsTree

A jsTree is used by the `userdata`, `usertags` and `public` pages to browse hierarchical data. Each time a node is expanded, it uses appropriate AJAX calls to load children as json data. Further POST or DELETE AJAX calls are made to modify the data hierarchy by creating or deleting links between objects.

Selection changes in the jsTree and centre panel thumbnails cause events to be triggered by jQuery on the `$("#body")` element of the page, allowing other scripts to listen for these events. These are used to load selected data into the centre and right panels. The HTML for these panels contains additional scripts that also run when they load to setup their own event listeners.

There is also a global `update_thumbnails_panel` function that can be called by any script that needs to refresh the centre panel. For example, when the jsTree is used to add or remove Images from a selected Dataset.

Switching Groups and Users

The current group and user are stored in the [HTTP session](#) as `request.session['active_group']` and `request.session['user_id']`. These are used to define the data that is loaded in the main `userdata` and `usertags` pages.

The group and user are switched by the Groups and Users menu, which updates the session and reloads the page.

Show queries

Data in OMERO can be linked from the webclient with URLs of the form `/webclient/?show=image-23|image-34`

In the `load_template` view, the first object is queried from OMERO and its parent containers are also loaded. The owner and group of the top container is used to set the `active_group` and `user_id` so that the main page loads the appropriate data hierarchy.

The jsTree does its own lookup from the query string, retrieving json data and using this to expand the appropriate nodes to show the specified objects.

Javascript code

The majority of javascript code is jQuery-based code that is embedded within the HTML templates and is run when the page loads.

Additional code is in static scripts, with functions generally name-spaced under an OME module.

Reusing OMERO sessions

When integrating other applications with OMERO.web you may want to automatically log in to OMERO.web using an existing OMERO session key. The session can be passed as a query parameter. For example a direct link to image will look as follows:

```
https://your_host/webgateway/img_detail/IMAGE_ID/?server=SERVER_ID&bssession=OMERO_
↪SESSION_KEY
```

This provides full access to OMERO.web, in the same way that logging in with a username and password would. It is therefore unsuited for giving others temporary access to data.

Note: The *SERVER_ID* should match the index from the list set using `omero.web.server_list` from the server session you created. If your list contains only one server, the index will be 1.

For more details about how to create an OMERO session see [server-side session](#) or use the [command line interface](#) to create one.

3.6 Insight

Note: With the release of OMERO 5.3.0, the OMERO.insight desktop client has entered **maintenance mode**, meaning it will only be updated if a major bug is discovered. Instead, the OME team will be focusing on developing the web clients. As a result, coding against this client is no longer recommended. Technical documentation can be found at <https://omero-insight.readthedocs.io/en/latest/>.

3.7 More on API Usage

OMERO can be extended by modifying these clients or by writing your own in any of the supported languages.

3.7.1 Developing OMERO clients

Note:

- If you are only interested in **using** our OMERO clients, please see the [OMERO clients overview](#) section, which will point you to user guides, demo videos, and download sites.
- This page is intended for developers already familiar with client/server programming. If you are not, **your best starting point is to read the [Hello World chapter of the Ice manual](#) (or more)**. A deeper understanding of Ice might not be necessary, but certainly understanding the Ice basics will make reading this guide **much** easier.

For developers, there are many examples listed below, all of which are stored under: <https://github.com/ome/openmicroscopy/tree/develop/examples> and buildable/runnable via `scons`:

```
cd omero-src
./build.py build-all
cd omero-src/examples
python ../target/scons/scons.py
```

Other examples (in Python) can be found [here](#).

Introduction

A Blitz client is any application which uses the *OMERO Application Programming Interface* to talk to the *OMERO.blitz* server in any of the supported languages, like *Python*, *C++*, *Java*, or *MATLAB*. A general understanding of the *OMERO.server overview* may make what is happening behind the scenes more transparent, but is not necessary. The points below outline all that an application writer is expected to know with links to further information where necessary.

Distributed computing

The first hurdle when beginning to work with OMERO is to realize that building distributed-object systems is different from both building standalone clients and writing web applications in frameworks like `mod_perl`, `django`, or `Ruby on Rails`. The remoting framework used by OMERO is *Ice* from ZeroC. Ice is comparable to CORBA in many ways, but is typically easier to use.

A good first step is to be aware of the difference between remote and local invocations. Any invocation on a proxy (`<class_name>Prx`, described below) will result in a call over the network with all the costs that entails. The often-cited *fallacies of distributed computing* all apply, and the developer must be aware of concurrency and latency issues, as well as complete loss of connectivity, all of which we will discuss below.

Objects

Before we can begin talking about what you can do with OMERO (the remote method calls available in the *OMERO Application Programming Interface*), it is helpful to first know what the objects are that we will be distributing. These are the only types that can pass through the API.

“Slice” mapping language

Ice provides an *interface definition language (IDL)* for defining class hierarchies for passing data in a binary format. Similar to WSDL in web services or CORBA’s IDL, slice provides a way to specify how types can pass between different programming languages. For just that reason, several constructs not available in all the supported languages are omitted:

- multiple inheritance (C++ and Python)
- nullable primitive wrappers (e.g. Java’s `java.lang.Integer`)
- interfaces (Java)
- `HashSet` types
- iterator types

Primitives

Slice defines the usual primitives – `long`, `string`, `bool`, as well as `int`, `double`, and `float` – which map into each language as would be expected. Aliases like “`Ice::Long`” are available for C++ to handle both 32 and 64 bit architectures.

A simple struct can then be built out of any combination of these types. From <https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/slice/omero/System.ice>:

```
// The EventContext is all the information the server knows about a
// given method call, including user, read/write status, etc.
class EventContext
{
    ...
    long    userId;
    string  userName;
    ...
    bool    isAdmin;
    ...
}
```

Sequences, dictionaries, enums, and constants

Other than the “user-defined classes” which we will get to below, slice provides only four built-in building blocks for creating a type hierarchy.

- **Sequences. & Dictionaries** : Most of the sequences and dictionaries in use by the *OMERO Application Programming Interface* are defined in <https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/slice/omero/Collections.ice>. Each sequence or dictionary must be defined before it can be used in any class. By default a sequence will map to an array of the given type in Java or a vector in C++, but these mappings can be changed via metadata. (In most cases, a `List` is used in the Java mapping).
- **Constants.** : Most of the enumerations for *OMERO Application Programming Interface* are defined in <https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/slice/omero/Constants.ice>. These are values which can be defined once and then referenced in each of the supported programming languages. The only real surprise when working with enumerations is that in Java each constant is mapped to an interface with a single public `final` static field named “value”.

```
#include <iostream>
#include <omero/Constants.h>
using namespace omero::constants;
int main() {
    std::cout << "By default, no method call can pass more than ";
    std::cout << MESSAGE_SIZE_MAX << "kb" << std::endl;
    std::cout << "By default, client.createSession() will wait ";
    std::cout << (CONNECT_TIMEOUT / 1000) << " seconds for a connection" << std::endl;
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/constants.cpp>

```
sz=omero.constants.MESSAGE_SIZE_MAX.value;
to=omero.constants.CONNECT_TIMEOUT.value/1000;
disp(sprintf('By default, no method call can pass more than %d kb',sz));
disp(sprintf('By default, client.createSession() will wait %d seconds for a connection',
↳to));
```


Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/constants.m>

```
from omero.constants import *
print("By default, no method call can pass more than %s kb" % MESSAGE_SIZE_MAX)
print("By default, client.createSession() will wait %s seconds for a connection" %
      (CONNECT_TIMEOUT/1000))
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/constants.py>

```
import static omero.rtypes.*;
public class constants {
    public static void main(String[] args) {
        System.out.println(String.format(
            "By default, no method call can pass more than %s kb",
            omero.constants.MESSAGE_SIZE_MAX.value));
        System.out.println(String.format(
            "By default, client.createSession() will wait %s seconds for a connection",
            omero.constants.CONNECT_TIMEOUT.value/1000));
    }
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/constants.java>

- **Enums.** Finally, enumerations which are less used through *OMERO Application Programming Interface*, but which can be useful for simplifying working with constants.

```
#include <iostream>
#include <omero/Constants.h>
using namespace omero::constants::projection;
int main() {
    std::cout << "IProjection takes arguments of the form: ";
    std::cout << MAXIMUM_INTENSITY;
    std::cout << std::endl;
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/enumerations.cpp>

```
v=omero.constants.projection.ProjectionType.MAXIMUM_INTENSITY.value();
disp(sprintf('IProjection takes arguments of the form: %s', v));
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/enumerations.m>

```
import omero
import omero_Constants_ice
print("IProjection takes arguments of the form: %s" % omero.constants.projection.
      ProjectionType.MAXIMUM_INTENSITY)
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/enumerations.py>

```
public class enumerations {
    public static void main(String[] args) {
        System.out.println(String.format(
            "IProjection takes arguments of the form: %s",
            omero.constants.projection.ProjectionType.MAXIMUM_INTENSITY));
    }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/enumerations.java>

RTypes

In Java, the Ice primitives map to non-nullable primitives. And in fact, for the still nullable types `java.lang.String` as well as all collections and arrays, Ice goes so far as to send an empty string (`""`) or `collection([])` rather than null.

However, the database and OMERO support nullable values and so *OMERO.blitz* defines a hierarchy of types which wraps the primitives: *RTypes*. Since Ice allows references to be nulled, as opposed to primitives, it is possible to send null strings, integers, etc.

```
#include <omero/RTypesI.h>
using namespace omero::rtypes;
int main() {
    omero::RStringPtr s = rstring("value");
    omero::RBoolPtr b = rbool(true);
    omero::RLongPtr l = rlong(1);
    omero::RIntPtr i = rint(1);
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/primitives.cpp>

```
import omero.rtypes;
a = rtypes.rstring('value');
b = rtypes.rbool(true);
l = rtypes.rlong(1);
i = rtypes.rint(1);
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/primitives.m>

```
from omero.rtypes import *
s = rstring("value")
b = rbool(True)
l = rlong(1)
i = rint(1)
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/primitives.py>

```
import static omero.rtypes.*;
public class primitives {
    public static void main(String[] args) {
        omero.RString a = rstring("value");
        omero.RBool b = rbool(true);
        omero.RLong l = rlong(1);
        omero.RInt i = rint(1);
    }
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/primitives.java>

The same works for collections. The *RCollection* subclass of *RType* holds a sequence of any other *RType*.

```
#include <omero/RTypesI.h>
using namespace omero::rtypes;
int main() {
    // Sets and Lists may be interpreted differently on the server
    omero::RListPtr l = rlist(); // rstring("a"), rstring("b"));
    omero::RSetPtr s = rset();   // rint(1), rint(2));
                                // No-varargs (#1242)
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/rcollection.cpp>

```
% Sets and Lists may be interpreted differently on the server
ja = javaArray('omero.RString',2);
ja(1) = omero.rtypes.rstring('a');
ja(2) = omero.rtypes.rstring('b');
list = omero.rtypes.rlist(ja)
ja = javaArray('omero.RInt',2);
ja(1) = omero.rtypes.rint(1);
ja(2) = omero.rtypes.rint(2);
set = omero.rtypes.rset(ja)
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/rcollection.m>

```
import omero
from omero.rtypes import *
# Sets and Lists may be interpreted differently on the server
list = rlist(rstring("a"), rstring("b"));
set = rset(rint(1), rint(2));
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/rcollection.py>

```
import static omero.rtypes.*;
public class rcollection {
    public static void main(String[] args) {
        // Sets and Lists may be interpreted differently on the server
        omero.RList list = rlist(rstring("a"), rstring("b"));
        omero.RSet set = rset(rint(1), rint(2));
    }
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/rcollection.java>

A further benefit of the RTypes is that they support **polymorphism**. The original *OMERO Application Programming Interface* was designed strictly for Java, in which the `java.lang.Object` type or collections of `java.lang.Object` could be passed. This is not possible with Ice, since there is no `Any` type as there is in CORBA.

Instead, `omero.RType` is the abstract superclass of our “remote **type**” hierarchy, and any method which takes an “RType” can take any subclass of “RType”.

To allow other types discussed later to also take part in the polymorphism, it is necessary to include RType wrappers for them. This is the category that `omero::RObject` and `omero::RMap` fall into.

`omero::RTime` and `omero::RClass` fall into a different category. They are identical to `omero::RLong` and `omero::RString`, respectively, but are provided as type safe variants.

OMERO model objects

With these components – rtypes, primitives, constants, etc. – it is possible to define the core nouns of OME, the *OME-Remote Objects*. The OMERO *OME-Remote Objects* is a translation of the *OME XML specification* into objects for use by the server, built out of RTypes, sequences and dictionaries, and Details.

Details

The `omero.model.Details` object contains security and other internal information which does not contain any domain value. Attempting to set any values which are not permitted, will result in a `SecurityViolation`, for example trying to change the `details.owner` to the current user.

```
#include <omero/model/ImageI.h>
#include <omero/model/PermissionsI.h>
using namespace omero::model;
int main() {
    ImagePtr image = new ImageI();
    DetailsPtr details = image->getDetails();
    PermissionsPtr p = new PermissionsI();
    p->setUserRead(true);
    assert(p->isUserRead());
    details->setPermissions(p);
    // Available when returned from server
    // Possibly modifiable
    details->getOwner();
    details->setGroup(new ExperimenterGroupI(1L, false));
    // Available when returned from server
    // Not modifiable
    details->getCreationEvent();
    details->getUpdateEvent();
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/details.cpp>

```
image = omero.model.ImageI();
details_ = image.getDetails();
p = omero.model.PermissionsI();
p.setUserRead(true);
assert( p.isUserRead() );
details_.setPermissions( p );
% Available when returned from server
% Possibly modifiable
details_.getOwner();
details_.setGroup( omero.model.ExperimenterGroupI(1, false) );
% Available when returned from server
% Not modifiable
details_.getCreationEvent();
details_.getUpdateEvent();
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/details.m>

```
import omero
import omero.clients
```

(continues on next page)

(continued from previous page)

```

image = omero.model.ImageI()
details = image.getDetails()
p = omero.model.PermissionsI()
p.setUserRead(True)
assert p.isUserRead()
details.setPermissions(p)
# Available when returned from server
# Possibly modifiable
details.getOwner()
details.setGroup(omero.model.ExperimenterGroupI(1L, False))
# Available when returned from server
# Not modifiable
details.getCreationEvent()
details.getUpdateEvent()

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/details.py>

```

import omero.model.Image;
import omero.model.ImageI;
import omero.model.Details;
import omero.model.Permissions;
import omero.model.PermissionsI;
import omero.model.ExperimenterGroupI;
public class details {
    public static void main(String args[]) {
        Image image = new ImageI();
        Details details = image.getDetails();
        Permissions p = new PermissionsI();
        p.setUserRead(true);
        assert p.isUserRead();
        details.setPermissions(p);
        // Available when returned from server
        // Possibly modifiable
        details.getOwner();
        details.setGroup(new ExperimenterGroupI(1L, false));
        // Available when returned from server
        // Not modifiable
        details.getCreationEvent();
        details.getUpdateEvent();
    }
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/details.java>

Warning: Do not use *IQuery*'s `projection` operation to read a data object *obj*'s `obj.details.permissions` field because it can give a misleading result. Instead *OMERO.web* instantiates a Map in reading `obj_details_permissions`. This pattern is shown in the first section of OME's *Hibernate 3.5 Training* where it covers the querying of permissions.

ObjectFactory and casting

In the previous examples, you may have noticed how there are two classes for each type: `Image` and `ImageI`. Classes defined in slice are by default data objects, more like C++’s `structs` than anything else. As soon as a class defines a method, however, it becomes an abstract entity and requires application writers to provide a **concrete implementation** (hence the “I”). All OMERO classes define methods, but OMERO takes care of providing the implementations for you via code generation. For each slice-defined and Ice-generated class `omero.model.Something`, there is an OMERO-generated class `omero.model.SomethingI` which can be instantiated.

```
#include <omero/model/ImageI.h>
#include <omero/model/DatasetI.h>
using namespace omero::model;
int main() {
    ImagePtr image = new ImageI();
    DatasetPtr dataset = new DatasetI(1L, false);
    image->linkDataset(dataset);
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/constructors.cpp>

```
import omero.model.*;
image = ImageI();
dataset = DatasetI(1, false);
image.linkDataset(dataset)
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/constructors.m>

```
import omero
import omero.clients
image = omero.model.ImageI()
dataset = omero.model.DatasetI(long(1), False)
image.linkDataset(dataset)
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/constructors.py>

```
import java.util.Iterator;
import omero.model.Image;
import omero.model.ImageI;
import omero.model.Dataset;
import omero.model.DatasetI;
import omero.model.DatasetImageLink;
import omero.model.DatasetImageLinkI;
public class constructors {
    public static void main(String args[]) {
        Image image = new ImageI();
        Dataset dataset = new DatasetI(1L, false);
        image.linkDataset(dataset);
    }
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/constructors.java>

When *OME-Remote Objects* instances are serialized over the wire and arrive in the client, the Ice runtime must determine which constructor to call. It consults with the ObjectFactory, also provided by OMERO, to create the new classes.

If you would like to have your own classes or subclasses created on deserialization, see the **Advanced topics** section below.

Such concrete implementations provide features which are not available in the solely Ice-based versions. When you would like to use these features, it is necessary to down-cast to the OMERO-based type.

For example, objects in each language binding provide a “more natural” form of iteration for that language.

```
#include <omero/model/ImageI.h>
#include <omero/model/DatasetI.h>
#include <omero/model/DatasetImageLinkI.h>
using namespace omero::model;
int main() {
    ImageIPtr image = new ImageI();
    DatasetIPtr dataset = new DatasetI();
    DatasetImageLinkPtr link = dataset->linkImage(image);
    omero::model::ImageDatasetLinksSeq seq = image->copyDatasetLinks();
    ImageDatasetLinksSeq::iterator beg = seq.begin();
    while(beg != seq.end()) {
        beg++;
    }
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/iterators.cpp>

```
import omero.model.*;
image = ImageI();
dataset = DatasetI();
link = dataset.linkImage(image);
it = image.iterateDatasetLinks();
while it.hasNext()
    it.next().getChild().getName()
end
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/iterators.m>

```
import omero
from omero_model_ImageI import ImageI
from omero_model_DatasetI import DatasetI
from omero_model_DatasetImageLinkI import DatasetImageLinkI
image = ImageI()
dataset = DatasetI()
link = dataset.linkImage(image)
for link in image.iterateDatasetLinks():
    link.getChild().getName();
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/iterators.py>

```
import omero.model.ImageI;
import omero.model.Dataset;
import omero.model.DatasetI;
import omero.model.DatasetImageLink;
import omero.model.DatasetImageLinkI;
import java.util.*;
public class iterators {
```

(continues on next page)

(continued from previous page)

```

public static void main(String args[]) {
    ImageI image = new ImageI();
    Dataset dataset = new DatasetI();
    DatasetImageLink link = dataset.linkImage(image);
    Iterator<DatasetImageLinkI> it = image.iterateDatasetLinks();
    while (it.hasNext()) {
        it.next().getChild().getName();
    }
}
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/iterators.java>

]

Also, each concrete implementation provides static constants of various forms.

```

#include <omero/model/ImageI.h>
#include <iostream>
int main() {
    std::cout << omero::model::ImageI::NAME << std::endl;
    std::cout << omero::model::ImageI::DATASETLINKS << std::endl;
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/staticfields.cpp>

```

disp(omero.model.ImageI.NAME);
disp(omero.model.ImageI.DATASETLINKS);

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/staticfields.m>

```

import omero
from omero_model_ImageI import ImageI as ImageI
print(ImageI.NAME)
print(ImageI.DATASETLINKS)

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/staticfields.py>

```

import omero.model.ImageI;
public class staticfields {
    public static void main(String[] args) {
        System.out.println(ImageI.NAME);
        System.out.println(ImageI.DATASETLINKS);
    }
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/staticfields.java>

Visibility and loadedness

In the constructor example above, a constructor with two arguments was used to create the `Dataset` instance linked to the new `Image`. The `Dataset` instance so created is considered “unloaded”.

Objects and collections can be created unloaded as a pointer to an actual instance or they may be returned unloaded from the server when they are not actively accessed in a query. Because of the interconnectedness of the *OME-Remote Objects*, loading one object could conceivably require downloading a large part of the database if there were not some way to “snip-off” sections.

```
#include <omero/model/ImageI.h>
#include <omero/model/DatasetI.h>
#include <omero/ClientErrors.h>
using namespace omero::model;
int main() {
    ImagePtr image = new ImageI();           // A loaded object by default
    assert(image->isLoaded());
    image->unload();                          // can then be unloaded
    assert(! image->isLoaded());
    image = new ImageI( 1L, false );         // Creates an unloaded "proxy"
    assert(! image->isLoaded());
    image->getId();                           // Ok
    try {
        image->getName();                    // No data access is allowed other than id.
    } catch (const omero::ClientError& ce) {
        // Ok.
    }
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/unloaded.cpp>

```
image = omero.model.ImageI();               % A loaded object by default
assert(image.isLoaded());
image.unload();                             % can then be unloaded
assert( ~ image.isLoaded() );
image = omero.model.ImageI( 1, false );     % Creates an unloaded "proxy"
assert( ~ image.isLoaded() );
image.getId();                              % Ok.
try
    image.getName();                         % No data access is allowed other than id
catch ME
    % OK
end
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/unloaded.m>

```
import omero
import omero.clients
image = omero.model.ImageI()                # A loaded object by default
assert image.isLoaded()
image.unload()                              # can then be unloaded
assert (not image.isLoaded())
image = omero.model.ImageI( 1L, False )     # Creates an unloaded "proxy"
assert (not image.isLoaded())
```

(continues on next page)

(continued from previous page)

```

image.getId()                                # Ok
try:
    image.getName()                          # No data access is allowed other than id.
except:
    pass

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/unloaded.py>

```

import omero.model.ImageI;
public class unloaded {
    public static void main(String args[]) {
        ImageI image = new ImageI();           // A loaded object by default
        assert image.isLoaded();
        image.unload();                       // can then be unloaded
        assert ! image.isLoaded();
        image = new ImageI( 1L, false );       // Creates an unloaded "proxy"
        assert ! image.isLoaded();
        image.getId();                         // Ok.
        try {
            image.getName();                   // No data access is allowed other than
↪ id.
        } catch (Exception e) {
            // Ok.
        }
    }
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/unloaded.java>

When saving objects that have unloaded instances in their graph, the server will automatically fill in the values. So, if your `Dataset` contains a collection of `Images`, all of which are unloaded, then they will be reloaded before saving, based on the id. If, however, you had tried to set a value on one of the `Images`, you will get an exception.

To prevent errors when working with unloaded objects, all the *OME-Remote Objects* classes are marked as protected in the slice definitions which causes the implementations in each language to try to hide the fields. In Java and C++ this results in fields with “protected” visibility. In Python, an underscore is prefixed to all the variables. (In the Python case, we have also tried to “strengthen” the hiding of the fields, by overriding `__setattr__`. This is not full proof, but only so much can be done to hide values in Python.)

Collections

Just as an entire object can be unloaded, any collection field can also be unloaded. However, as mentioned above, since it is not possible to send a null collection over the wire with Ice and working with `RTypes` can be inefficient, all the *OME-Remote Objects* collections are hidden behind several methods.

```

#include <omero/model/DatasetI.h>
#include <omero/model/DatasetImageLinkI.h>
#include <omero/model/EventI.h>
#include <omero/model/ImageI.h>
#include <omero/model/PixelsI.h>
using namespace omero::model;
int main(int argc, char* argv[]) {

```

(continues on next page)

(continued from previous page)

```

ImagePtr image = new ImageI(1L, true);
image->getDetails()->setUpdateEvent( new EventI(1L, false) );
// On creation, all collections are
// initialized to empty, and can be added
// to.
assert(image->sizeOfDatasetLinks() == 0);
DatasetPtr dataset = new DatasetI(1L, false);
DatasetImageLinkPtr link = image->linkDataset(dataset);
assert(image->sizeOfDatasetLinks() == 1);
// If you want to work with this collection,
// you'll need to get a copy.
ImageDatasetLinksSeq links = image->copyDatasetLinks();
// When you are done working with it, you can
// unload the datasets, assuming the changes
// have been persisted to the server.
image->unloadDatasetLinks();
assert(image->sizeOfDatasetLinks() < 0);
try {
    image->linkDataset( new DatasetI() );
} catch (...) {
    // Can't access an unloaded collection
}
// The reload...() method allows one instance
// to take over a collection from another, if it
// has been properly initialized on the server.
// sameImage will have its collection unloaded.
ImagePtr sameImage = new ImageI(1L, true);
sameImage->getDetails()->setUpdateEvent( new EventI(1L, false) );
sameImage->linkDataset( new DatasetI(1L, false) );
image->reloadDatasetLinks( sameImage );
assert(image->sizeOfDatasetLinks() == 1);
assert(sameImage->sizeOfDatasetLinks() < 0);
// If you would like to remove all the member
// elements from a collection, don't unload it
// but "clear" it.
image->clearDatasetLinks();
// Saving this to the database will remove
// all dataset links!
// Finally, all collections can be unloaded
// to use an instance as a single row in the database.
image->unloadCollections();
// Ordered collections have slightly different methods.
image = new ImageI(1L, true);
image->addPixels( new PixelsI() );
image->getPixels(0);
image->getPrimaryPixels(); // Same thing
image->removePixels( image->getPixels(0) );
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/collectionmethods.cpp>

```
import omero.model.*;
```

(continues on next page)

(continued from previous page)

```

image = ImageI(1, true);
image.getDetails().setUpdateEvent( EventI(1, false) );
% On creation, all collections are
% initialized to empty, and can be added
% to.
assert(image.sizeOfDatasetLinks() == 0);
dataset = DatasetI(1, false);
link = image.linkDataset(dataset);
assert(image.sizeOfDatasetLinks() == 1);
% If you want to work with this collection,
% you'll need to get a copy.
links = image.copyDatasetLinks();
% When you are done working with it, you can
% unload the datasets, assuming the changes
% have been persisted to the server.
image.unloadDatasetLinks();
assert(image.sizeOfDatasetLinks() < 0);
try
    image.linkDataset( DatasetI() );
catch ME
    % Can't access an unloaded collection
end
% The reload...() method allows one instance
% to take over a collection from another, if it
% has been properly initialized on the server.
% sameImage will have its collection unloaded.
sameImage = ImageI(1, true);
sameImage.getDetails().setUpdateEvent( EventI(1, false) );
sameImage.linkDataset( DatasetI(1, false) );
image.reloadDatasetLinks( sameImage );
assert(image.sizeOfDatasetLinks() == 1);
assert(sameImage.sizeOfDatasetLinks() < 0);
% If you would like to remove all the member
% elements from a collection, don't unload it
% but "clear" it.
image.clearDatasetLinks();
% Saving this to the database will remove
% all dataset links!
% Finally, all collections can be unloaded
% to use an instance as a single row in the database.
image.unloadCollections();
% Ordered collections have slightly different methods.
image = ImageI(1, true);
image.addPixels( PixelsI() );
image.getPixels(0);
image.getPrimaryPixels(); % Same thing
image.removePixels( image.getPixels(0) );

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/collectionmethods.m>

```

import omero
import omero.clients

```

(continues on next page)

(continued from previous page)

```

ImageI = omero.model.ImageI
DatasetI = omero.model.DatasetI
EventI = omero.model.EventI
PixelsI = omero.model.PixelsI
image = ImageI(long(1), True)
image.getDetails().setUpdateEvent( EventI(1L, False) )
# On creation, all collections are
# initialized to empty, and can be added
# to.
assert image.sizeOfDatasetLinks() == 0
dataset = DatasetI(long(1), False)
link = image.linkDataset(dataset)
assert image.sizeOfDatasetLinks() == 1
# If you want to work with this collection,
# you'll need to get a copy.
links = image.copyDatasetLinks()
# When you are done working with it, you can
# unload the datasets, assuming the changes
# have been persisted to the server.
image.unloadDatasetLinks()
assert image.sizeOfDatasetLinks() < 0
try:
    image.linkDataset( DatasetI() )
except:
    # Can't access an unloaded collection
    pass
# The reload...() method allows one instance
# to take over a collection from another, if it
# has been properly initialized on the server.
# sameImage will have its collection unloaded.
sameImage = ImageI(1L, True)
sameImage.getDetails().setUpdateEvent( EventI(1L, False) )
sameImage.linkDataset( DatasetI(long(1), False) )
image.reloadDatasetLinks( sameImage )
assert image.sizeOfDatasetLinks() == 1
assert sameImage.sizeOfDatasetLinks() < 0
# If you would like to remove all the member
# elements from a collection, don't unload it
# but "clear" it.
image.clearDatasetLinks()
# Saving this to the database will remove
# all dataset links!
# Finally, all collections can be unloaded
# to use an instance as a single row in the database.
image.unloadCollections()
# Ordered collections have slightly different methods.
image = ImageI(long(1), True)
image.addPixels( PixelsI() )
image.getPixels(0)
image.getPrimaryPixels() # Same thing
image.removePixels( image.getPixels(0) )

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/collectionmethods.py>

```

import omero.model.Dataset;
import omero.model.DatasetI;
import omero.model.DatasetImageLink;
import omero.model.DatasetImageLinkI;
import omero.model.EventI;
import omero.model.Image;
import omero.model.ImageI;
import omero.model.Pixels;
import omero.model.PixelsI;
import java.util.*;

public class collectionmethods {
    public static void main(String args[]) {
        Image image = new ImageI(1, true);
        image.getDetails().setUpdateEvent( new EventI(1L, false) );
        // On creation, all collections are
        // initialized to empty, and can be added
        // to.
        assert image.sizeOfDatasetLinks() == 0;
        Dataset dataset = new DatasetI(1L, false);
        DatasetImageLink link = image.linkDataset(dataset);
        assert image.sizeOfDatasetLinks() == 1;
        // If you want to work with this collection,
        // you'll need to get a copy.
        List<DatasetImageLink> links = image.copyDatasetLinks();
        // When you are done working with it, you can
        // unload the datasets, assuming the changes
        // have been persisted to the server.
        image.unloadDatasetLinks();
        assert image.sizeOfDatasetLinks() < 0;
        try {
            image.linkDataset( new DatasetI() );
        } catch (Exception e) {
            // Can't access an unloaded collection
        }
        // The reload...() method allows one instance
        // to take over a collection from another, if it
        // has been properly initialized on the server.
        // sameImage will have its collection unloaded.
        Image sameImage = new ImageI(1L, true);
        sameImage.getDetails().setUpdateEvent( new EventI(1L, false) );
        sameImage.linkDataset( new DatasetI(1L, false) );
        image.reloadDatasetLinks( sameImage );
        assert image.sizeOfDatasetLinks() == 1;
        assert sameImage.sizeOfDatasetLinks() < 0;
        // If you would like to remove all the member
        // elements from a collection, don't unload it
        // but "clear" it.
        image.clearDatasetLinks();
        // Saving this to the database will remove
        // all dataset links!
        // Finally, all collections can be unloaded
        // to use an instance as a single row in the database.
        image.unloadCollections();
    }
}

```

(continues on next page)

(continued from previous page)

```

// Ordered collections have slightly different methods.
image = new ImageI(1L, true);
image.addPixels( new PixelsI() );
image.getPixels(0);
image.getPrimaryPixels(); // Same thing
image.removePixels( image.getPixels(0) );
}
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/collectionmethods.java>

These methods prevent clients from accessing the collections directly, and any improper access will lead to an `omero.ClientError`.

Interfaces

As mentioned above, one of the Java features which is missing from the slice definition language is the ability to have concrete classes implement **multiple** interfaces. Much of the *OME-Remote Objects* in the RMI-based types (`ome.model`) was based on the use of interfaces.

- `:model_source: IObject <src/main/java/ome/model/IObject.java>` is the root interface for all object types. **Methods:** `getId()`, `getDetails()`, ...
- `:model_source: IEnum <src/main/java/ome/model/IEnum.java>` is an enumeration value. **Methods:** `getValue()`
- `:model_source: ILink <src/main/java/ome/model/ILink.java>` is a link between two other types. **Methods:** `getParent()`, `getChild()`
- `:model_source: IMutable <src/main/java/ome/model/IMutable.java>` is an instance for changes will be persisted. **Methods:** `getVersion()`

Instead, the Ice-based types (`omero.model`) all subclass from the same concrete type – `omero.model.IObject` – and it has several methods defined for testing which of the `ome.model` interfaces are implemented by any type.

Use of such methods is naturally less object-oriented and requires if/then blocks, but within the confines of the mapping language is a next-best option.

No cpp example

```

import omero.model.*;
o = EventI();
assert( ~ o.isMutable() );
o = ExperimenterI();
assert( o.isMutable() );
assert( o.isGlobal() );
assert( o.isAnnotated() );
o = GroupExperimenterMapI();
assert( o.isLink() );
someObject = ExperimenterI();
% Some method call and you no longer know what someObject is
if ( ~ someObject.isMutable() )
    % No need to update
elseif (someObject.isAnnotated())

```

(continues on next page)

(continued from previous page)

```
% deleteAnnotations(someObject);
end
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/interfaces.m>

```
import omero
from omero_model_EventI import EventI
from omero_model_ExperimenterI import ExperimenterI
from omero_model_GroupExperimenterMapI import GroupExperimenterMapI
assert ( not EventI().isMutable() )
assert ExperimenterI().isMutable()
assert ExperimenterI().isGlobal()
assert ExperimenterI().isAnnotated()
assert GroupExperimenterMapI().isLink()
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/interfaces.py>

```
import omero.model.IObject;
import omero.model.EventI;
import omero.model.ExperimenterI;
import omero.model.GroupExperimenterMapI;
public class interfaces {
    public static void main(String args[]) {
        assert ! new EventI().isMutable();
        assert new ExperimenterI().isMutable();
        assert new ExperimenterI().isGlobal();
        assert new ExperimenterI().isAnnotated();
        assert new GroupExperimenterMapI().isLink();
        IObject someObject = new ExperimenterI();
        // Some method call and you no longer know what someObject is
        if ( ! someObject.isMutable() ) {
            // No need to update
        } else if (someObject.isAnnotated()) {
            // deleteAnnotations(someObject);
        }
    }
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/interfaces.java>

Improvement of this situation by adding abstract classes is planned. However, the entire functionality will not be achievable because of single inheritance.

Language-specific behavior

Smart pointers (C++ only)

An important consideration when working with C++ is that the *OME-Remote Objects* classes themselves have no copy-constructors and no assignment operator (operator=), and so cannot be allocated on the stack. Combined with smart pointers this effectively prevents memory leaks.

The code generated types must be allocated on the heap with new and used in combination with the smart pointer typedefs which handle calling the destructor when the reference count hits zero.


```
#include <omero/model/ImageI.h>
using namespace omero::model;
int main()
{
    // ImageI image();           // ERROR
    // ImageI image = new ImageI(); // ERROR
    ImageIPtr image1 = new ImageI(); // OK
    ImageIPtr image2(new ImageI()); // OK
    // image1 pointer takes value of image2
    // image1's content is garbage collected
    image1 = image2;
    //
    // Careful with boolean contexts
    //
    if (image1 && image1 == 1) {
        // Means non-null
        // This object can be dereferenced
    }
    ImageIPtr nullImage; // No assignment
    if ( !nullImage && nullImage == 0) {
        // Dereferencing nullImage here would throw an exception:
        // nullImage->getId(); // IceUtil::NullHandleException !
    }
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/smartpointers.cpp>

No m example

No py example

No java example

Warning: As shown in the example, using a smart pointer instance in a boolean or integer/long context, returns 1 for true (i.e. non-null) or 0 for false (i.e. null). Be especially careful with the RTypes.

For more information, see [6.14.6 Smart Pointers for Classes](#) in the Ice manual, which also describes the `Ice.GC.Interval` parameter which determines how often garbage collection runs in C++ to reap objects. This is necessary with the *OME-Remote Objects* since there are inherently cycles in the object graph.

Another point type which may be of use is `omero::client_ptr`. It also performs reference counting and will call `client.closeSession()` once the reference count hits zero. Without `client_ptr`, your code will need to be surrounded by a try/catch block. Otherwise, 1) sessions will be left open on the server, and 2) your client may hang on exit.

```
#include <omero/client.h>
int main(int argc, char* argv[])
{
    // Duplicating the argument list. ticket:1246
    Ice::StringSeq args1 = Ice::argsToStringSeq(argc, argv);
    Ice::StringSeq args2(args1);
```

(continues on next page)

(continued from previous page)

```

Ice::InitializationData id1, id2;
id1.properties = Ice::createProperties(args1);
id2.properties = Ice::createProperties(args2);
// Either
omero::client client(id1);
try {
    // Do something like
    // client.createSession();
} catch (...) {
    client.closeSession();
}
//
// Or
//
{
    omero::client_ptr client = new omero::client(id2);
    // Do something like
    // client->createSession();
}
// Client was destroyed via RAII
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/clientpointer.cpp>

No m example

No py example

No java example

__getattr__ and __setattr__ (Python only)

Like smart pointers for *OMERO C++ language bindings*, the *OMERO Python language bindings* SDK defines `__getattr__` and `__setattr__` methods for all *OME-Remote Objects* classes. Rather than explicitly calling the `getFoo()` and `setFoo()` methods, field-like access can be used. (It should be noted, however, that the accessors will perform marginally faster.)

No cpp example

No m example

```

import omero
import omero.clients
from omero.rtypes import *
i = omero.model.ImageI()
#
# Without __getattr__ and __setattr__
#
i.setName( rstring("name") )
assert i.getName().getValue() == "name"

```

(continues on next page)

(continued from previous page)

```
#
# With __getattr__ and __setattr__
#
i = omero.model.ImageI()
i.name = rstring("name")
assert i.name.val == "name"
#
# Collections, however, cannot be accessed
# via the special methods due to the dangers
# outlined above
#
try:
    i.datasetLinks[0]
except AttributeError, ae:
    pass
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/getsetattr.py>

```
# No java example
```

Method inspection and code completion (MATLAB & Python)

Ice generates a number of internal (private) methods which are not intended for general consumption. Unfortunately, MATLAB's code-completion as well as Python's `dir` method return these methods, which can lead to confusion. In general, the API user can ignore any method beginning with an underscore or with `ice_`. For example,

```
>>>for i in dir(omero.model.ImageI):
...     if i.startswith("_") or i.startswith("ice_"):
...         print(i)
...
(snip)
_op_addAllDatasetImageLinkSet
_op_addAllImageAnnotationLinkSet
_op_addAllPixelsSet
_op_addAllRoiSet
_op_addAllWellSampleSet
...
ice_id
ice_ids
ice_isA
ice_ping
ice_postUnmarshal
ice_preMarshal
ice_staticId
ice_type
>>>
```

Services overview

After discussing the many types and how to create them, the next obvious question is what one can actually do with them. For that, we have to look at what services are provided by *OMERO.blitz*, how they are obtained, used, and cleaned up.

OMERO client configuration

The first step in accessing the *OMERO Application Programming Interface* and therefore the first thing to plan when writing an OMERO client is the proper configuration of an `omero.client` instance. The `omero.client` (or in C++ `omero::client`) class tries to wrap together and simplify as much of working with Ice as possible. Where it can, it imports or `<#includes>` types for you, creates an `Ice.Communicator` and registers an `ObjectFactory`. Typically, the only work on the client developers part is to properly configure the `omero.client` object and then login.

In the simplest case, configuration requires only the server host, username, and password with which you want to login. But as you can see below, there are various ways to configure your client, and this is really only the beginning.

```
#include <omero/client.h>
#include <iostream>
int main(int argc, char* argv[]) {
    // All configuration in file pointed to by
    // --Ice.Config=file.config
    // No username, password entered
    try {
        omero::client client1(argc, argv);
        client1.createSession();
        client1.closeSession();
    } catch (const Glacier2::PermissionDeniedException& pd) {
        // Bad password?
    } catch (const Ice::ConnectionRefusedException& cre) {
        // Bad address or port?
    }
    // Most basic configuration.
    // Uses default port 4064
    // createSession needs username and password
    try {
        omero::client client2("localhost");
        client2.createSession("root", "ome");
        client2.closeSession();
    } catch (const Glacier2::PermissionDeniedException& pd) {
        // Bad password?
    } catch (const Ice::ConnectionRefusedException& cre) {
        // Bad address or port?
    }
    // Configuration with port information
    try {
        omero::client client3("localhost", 24063);
        client3.createSession("root", "ome");
        client3.closeSession();
    } catch (const Glacier2::PermissionDeniedException& pd) {
        // Bad password?
    } catch (const Ice::ConnectionRefusedException& cre) {
        // Bad address or port?
    }
}
```

(continues on next page)

(continued from previous page)

```

}
// Advanced configuration in C++ takes place
// via an InitializationData instance.
try {
    Ice::InitializationData data;
    data.properties = Ice::createProperties();
    data.properties->setProperty("omero.host", "localhost");
    omero::client client4(data);
    client4.createSession("root", "ome");
    client4.closeSession();
} catch (const Glacier2::PermissionDeniedException& pd) {
    // Bad password?
} catch (const Ice::ConnectionRefusedException& cre) {
    // Bad address or port?
}
// std::map to be added (ticket:1278)
try {
    Ice::InitializationData data;
    data.properties = Ice::createProperties();
    data.properties->setProperty("omero.host", "localhost");
    data.properties->setProperty("omero.user", "root");
    data.properties->setProperty("omero.pass", "ome");
    omero::client client5(data);
    // Again, no username or password needed
    // since present in the data. But they *can*
    // be overridden.
    client5.createSession();
    client5.closeSession();
} catch (const Glacier2::PermissionDeniedException& pd) {
    // Bad password?
} catch (const Ice::ConnectionRefusedException& cre) {
    // Bad address or port?
}
}
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/configuration.cpp>

```

% All configuration in file pointed to by
% --Ice.Config=file.config
% No username, password entered
args = javaArray('java.lang.String',1);
args(1) = java.lang.String('--Ice.Config=ice.config');
client1 = omero.client(args);
client1.createSession();
client1.closeSession();
% Most basic configuration.
% Uses default port 4064
% createSession needs username and password
client2 = omero.client('localhost');
client2.createSession('root', 'ome');
client2.closeSession();
% Configuration with port information

```

(continues on next page)

(continued from previous page)

```

client3 = omero.client('localhost', 10463);
client3.createSession('root', 'ome');
client3.closeSession();
% Advanced configuration can also be done
% via an InitializationData instance.
data = Ice.InitializationData();
data.properties = Ice.Util.createProperties();
data.properties.setProperty('omero.host', 'localhost');
client4 = omero.client(data);
client4.createSession('root', 'ome');
client4.closeSession();
% Or alternatively via a java.util.Map instance
map = java.util.HashMap();
map.put('omero.host', 'localhost');
map.put('omero.user', 'root');
map.put('omero.pass', 'ome');
client5 = omero.client(map);
% Again, no username or password needed
% since present in the map. But they *can*
% be overridden.
client5.createSession();
client5.closeSession();

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/configuration.m>

```

import omero
import Ice
# All configuration in file pointed to by
# --Ice.Config=file.config or ICE_CONFIG
# environment variable;
# No username, password entered
try:
    client1 = omero.client()
    client1.createSession()
    client1.closeSession()
except Ice.ConnectionRefusedException:
    pass # Bad address or port?
# Most basic configuration.
# Uses default port 4064
# createSession needs username and password
try:
    client2 = omero.client("localhost")
    client2.createSession("root","ome")
    client2.closeSession()
except Ice.ConnectionRefusedException:
    pass # Bad address or port?
# Configuration with port information
try:
    client3 = omero.client("localhost", 24064)
    client3.createSession("root","ome")
    client3.closeSession()
except Ice.ConnectionRefusedException:

```

(continues on next page)

(continued from previous page)

```

    pass # Bad address or port?
# Advanced configuration can also be done
# via an InitializationData instance.
data = Ice.InitializationData()
data.properties = Ice.createProperties()
data.properties.setProperty("omero.host", "localhost")
try:
    client4 = omero.client(data)
    client4.createSession("root","ome")
    client4.closeSession()
except Ice.ConnectionRefusedException:
    pass # Bad address or port?
# Or alternatively via a dict instance
m = {"omero.host":"localhost",
     "omero.user":"root",
     "omero.pass":"ome"}
client5 = omero.client(m)
# Again, no username or password needed
# since present in the map. But they *can*
# be overridden.
try:
    client5.createSession()
    client5.closeSession()
except Ice.ConnectionRefusedException:
    pass # Bad address or port?

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/configuration.py>

```

public class configuration {
    public static void main(String[] args) throws Exception {
        // All configuration in file pointed to by
        // --Ice.Config=file.config
        // No username, password entered
        omero.client client1 = new omero.client(args);
        try {
            client1.createSession();
        } catch (Ice.ConnectionRefusedException cre) {
            // Bad address or port?
        } finally {
            client1.closeSession();
        }
        // Most basic configuration.
        // Uses default port 4064
        // createSession needs username and password
        omero.client client2 = new omero.client("localhost");
        try {
            client2.createSession("root", "ome");
        } catch (Ice.ConnectionRefusedException cre) {
            // Bad address or port?
        } finally {
            client2.closeSession();
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

// Configuration with port information
omero.client client3 = new omero.client("localhost", 24064);
try {
    client3.createSession("root", "ome");
} catch (Ice.ConnectionRefusedException cre) {
    // Bad address or port?
} finally {
    client3.closeSession();
}
// Advanced configuration can also be done
// via an InitializationData instance.
Ice.InitializationData data = new Ice.InitializationData();
data.properties = Ice.Util.createProperties();
data.properties.setProperty("omero.host", "localhost");
omero.client client4 = new omero.client(data);
try {
    client4.createSession("root", "ome");
} catch (Ice.ConnectionRefusedException cre) {
    // Bad address or port?
} finally {
    client4.closeSession();
}
// Or alternatively via a java.util.Map instance
java.util.Map<String, String> map = new java.util.HashMap<String, String>();
map.put("omero.host", "localhost");
map.put("omero.user", "root");
map.put("omero.pass", "ome");
omero.client client5 = new omero.client(map);
// Again, no username or password needed
// since present in the map. But they *can*
// be overridden.
try {
    client5.createSession();
} catch (Ice.ConnectionRefusedException cre) {
    // Bad address or port?
} finally {
    client5.closeSession();
}
}
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/configuration.java>

To find out more about using the `Ice.Config` file for configuration, see <https://github.com/ome/openmicroscopy/blob/develop/etc/templates/ice.config>.

What is a ServiceFactory?

In each of the examples above, the result of configuration was the ability to call `createSession` which returns a `ServiceFactoryPrx`.

The `ServiceFactory` is the clients representation of the user's *server-side session*, which multiple clients can connect to it simultaneously. A `ServiceFactoryPrx?` object is acquired on login via the `createSession` method, and persists until either it is closed or a timeout is encountered **unless** additional clients attach to it. This is done via `client.joinSession(String uuid)`. In that case, the session is not finally closed until its reference count drops to zero.

It produces services!

Once a client has been configured properly, and has an active in `ServiceFactory` in hand, it is time to start accessing services.

The collection of all services provided by OMERO is known as the *OMERO Application Programming Interface*. Each service is defined in a slice file under <https://github.com/ome/omero-blitz/tree/v5.7.2/src/main/slice/omero>. The central definitions are in <https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/slice/omero/API.ice>, along with the definition of `ServiceFactory` itself:

```
interface ServiceFactory extends Glacier2::Session
{
    // Central OMERO.blitz stateless services.
    IAdmin*      getAdminService() throws ServerError;
    IConfig*     getConfigService() throws ServerError;
    ...
    // Central OMERO.blitz stateful services.
    Gateway*     createGateway() throws ServerError;
    ...
}
```

In the definition above, the return values look like C/C++ pointers, which in Ice's definition language represents return-by-proxy. When a client calls, `serviceFactory.getAdminService()` it will receive an `IAdminPrx`. **Any call on that object is a remote invocation.**

Stateless vs. stateful

Most methods on the `ServiceFactory` return either a stateless or a stateful service factory. Stateless services are those returned by calls to "`getSomeNameService()`". They implement `omero.api.ServiceInterface` but not its subinterface `omero.api.StatefulServiceInterface`. Stateless services are for all intents and purposes singletons, though the implementation may vary.

Stateful services are returned by calls to "`createSomething()`" and implement `omero.api.StatefulServiceInterface`. Each maintains a state machine with varying rules on initialization and usage. It is important to guarantee that calls are ordered as described in the documentation for each stateful service. **It is also important to always close stateful services to free up server resources.** If you fail to manually call `StatefulServiceInterfacePrx.close()`, it will be called for you on session close/timeout.

What are timeouts?

The following code has a resource leak:

```
import omero, sys
c = omero.client()
s = c.createSession()
sys.exit(0)
```

Although the client will not suffer any consequences, this snippet leaves a *session* open on the server. If the server failed to eventually reap such sessions, they would eventually consume all available memory. To get around this, the server implements timeouts on all sessions. **It is the clients responsibility to periodically contact the server to keep the session alive.** Since threading policies vary in applications, no strict guideline is available on how to do this. Almost any API method will suffice to tell the server that the client is still active. Important is that the call happens within every timeout window.

No cpp example

No m example

```
import time
import omero
import threading
IDLETIME = 5
c = omero.client()
s = c.createSession()
re = s.createRenderingEngine()
class KeepAlive(threading.Thread):
    def run(self):
        self.stop = False
        while not self.stop:
            time.sleep(IDLETIME)
            print("calling keep alive")
            # Currently, passing a null or empty array to keepAllAlive
            # would suffice. For future-proofing, however, it makes sense
            # to pass stateful services.
            try:
                s.keepAllAlive([re])
            except:
                c.closeSession()
                raise
keepAlive = KeepAlive()
keepAlive.start()
time.sleep(IDLETIME * 2)
keepAlive.stop = True
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/timeout.py>

```
import omero.*;
import omero.api.*;
import omero.model.*;
import omero.sys.*;
public class timeout {
```

(continues on next page)

(continued from previous page)

```

static int IDLETIME = 5;
static client c;
static ServiceFactoryPrx s;
public static void main(String[] args) throws Exception {
    final int idletime = args.length > 1 ? Integer.parseInt(args[0]) : IDLETIME;
    c = new client(args);
    s = c.createSession();
    System.out.println(s.getAdminService().getEventContext().sessionUuid);
    final RenderingEnginePrx re = s.createRenderingEngine(); // for keep alive
    class Run extends Thread {
        public boolean stop = false;
        public void run() {
            while ( ! stop ) {
                try {
                    Thread.sleep(idletime*1000L);
                } catch (Exception e) {
                    // ok
                }
                System.out.println(System.currentTimeMillis() + " calling keep alive
→");
                try {
                    // Currently, passing a null or empty array to keepAllAlive
                    // would suffice. For future-proofing, however, it makes sense
                    // to pass stateful services.
                    s.keepAllAlive(new ServiceInterfacePrx[]{re});
                } catch (Exception e) {
                    c.closeSession();
                    throw new RuntimeException(e);
                }
            }
        }
    }
    final Run run = new Run();
    class Stop extends Thread {
        public void run() {
            run.stop = true;
        }
    }
    Runtime.getRuntime().addShutdownHook(new Stop());
    run.start();
}
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/timeout.java>

Exceptions

Probably the most critical thing to realize is that any call on a proxy, which includes `ServiceFactoryPrx` or any of the `*Prx` service classes is a remote invocation on the server. Therefore proper exception handling is critical. The definition of the various exceptions is outlined on the [Exception handling](#) page and so will not be repeated here. However, how are these sensibly used?

One easy rule is that every `omero.client` object which you successfully call `createSession()` on must have `closeSession()` called on it before you exit.

```
omero.client client = new omero.client();
client.createSession();
try {
    // do whatever you want
} finally {
    client.closeSession();
}
```

Obviously, the work you do in your client will be much more complicated, and may be under layers of application code. But when designing where active `omero.client` objects are kept, be sure that your clean-up code takes care of them.

IQuery

Now that we have a good idea of the basics, it might be interesting to start asking the server what it has got. The most powerful way of doing this is by using `IQuery` and the Hibernate Query Language (HQL).

```
#include <omero/api/IQuery.h>
#include <omero/client.h>
#include <omero/RTypesI.h>
#include <omero/sys/ParametersI.h>
using namespace omero::rtypes;
int main(int argc, char* argv[]) {
    omero::client_ptr client = new omero::client(argc, argv);
    omero::api::ServiceFactoryPrx sf = client->createSession();
    omero::api::IQueryPrx q = sf->getQueryService();
    std::string query_string = "select i from Image i where i.id = :id and name like _
↪:namedParameter";
    omero::sys::ParametersIPtr p = new omero::sys::ParametersI();
    p->add("id", rlong(1L));
    p->add("namedParameter", rstring("cell%mit%"));
    omero::api::IObjectList results = q->findAllByQuery(query_string, p);
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/queries.cpp>

```
[client,sf] = loadOmero;
try
    q = sf.getQueryService();
    query_string = 'select i from Image i where i.id = :id and name like :namedParameter
↪';
    p = omero.sys.ParametersI();
    p.add('id', omero.rtypes.rlong(1));
    p.add('namedParameter', omero.rtypes.rstring('cell%mit%'));
```

(continues on next page)

(continued from previous page)

```

    results = q.findAllByQuery(query_string, p) % java.util.List
catch ME
    client.closeSession();
end

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/queries.m>

```

import sys
import omero
from omero.rtypes import *
from omero_sys_ParametersI import ParametersI
client = omero.client(sys.argv)
try:
    sf = client.createSession()
    q = sf.getQueryService()
    query_string = "select i from Image i where i.id = :id and name like :namedParameter
↪";
    p = ParametersI()
    p.addId(1L)
    p.add("namedParameter", rstring("cell%mit%"));
    results = q.findAllByQuery(query_string, p)
finally:
    client.closeSession()

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/queries.py>

```

import java.util.List;
import static omero.rtypes.*;
import omero.api.ServiceFactoryPrx;
import omero.api.IQueryPrx;
import omero.model.IObject;
import omero.model.ImageI;
import omero.model.PixelsI;
import omero.sys.ParametersI;
public class queries {
    public static void main(String args[]) throws Exception {
        omero.client client = new omero.client(args);
        try {
            ServiceFactoryPrx sf = client.createSession();
            IQueryPrx q = sf.getQueryService();
            String query_string = "select i from Image i where i.id = :id and name like_
↪:namedParameter";
            ParametersI p = new ParametersI();
            p.add("id", rlong(1L));
            p.add("namedParameter", rstring("cell%mit%"));
            List<IObject> results = q.findAllByQuery(query_string, p);
        } finally {
            client.closeSession();
        }
    }
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/queries.java>

The `query_string` is an example of HQL. It looks a lot like SQL, but works with objects and fields rather than tables and columns (though in OMERO these are usually named the same). The `Parameters` object allow for setting named parameters (`:id`) in the query to allow for re-use, and is the only other argument need to `IQueryPrx.findAllByQuery()` to get a list of `IObject` instances back. They are guaranteed to be of type `omero::model::Image`, but you may have to cast them to make full use of that information.

IUpdate

After you have successfully read objects, an obvious thing to do is create your own. Below is a simple example of creating an image object:

```
#include <IceUtil/Time.h>
#include <omero/api/IUpdate.h>
#include <omero/client.h>
#include <omero/RTypesI.h>
#include <omero/model/ImageI.h>
using namespace omero::rtypes;
int main(int argc, char* argv[]) {
    omero::client_ptr client = new omero::client(argc, argv);
    omero::model::ImagePtr i = new omero::model::ImageI();
    i->setName( rstring("name") );
    i->setAcquisitionDate( rtime(IceUtil::Time::now().toMilliseconds()) );
    omero::api::ServiceFactoryPrx sf = client->createSession();
    omero::api::IUpdatePrx u = sf->getUpdateService();
    i = omero::model::ImagePtr::dynamicCast( u->saveAndReturnObject( i ) );
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/updates.cpp>

```
[client,sf] = loadOmero;
try
    i = omero.model.ImageI();
    i.setName(omero.rtypes.rstring('name'));
    i.setAcquisitionDate(omero.rtypes.rtime(java.lang.System.currentTimeMillis()));
    u = sf.getUpdateService();
    i = u.saveAndReturnObject( i );
    disp(i.getId().getValue());
catch ME
    disp(ME);
    client.closeSession();
end
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/updates.m>

```
import sys
import time
import omero
import omero.clients
from omero.rtypes import *
client = omero.client(sys.argv)
try:
    i = omero.model.ImageI()
    i.name = rstring("name")
```

(continues on next page)

(continued from previous page)

```

i.acquisitionDate = rtime(time.time() * 1000)
sf = client.createSession()
u = sf.getUpdateService()
i = u.saveAndReturnObject( i )
finally:
    client.closeSession()

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/updates.py>

```

import java.util.List;
import static omero.rtypes.*;
import omero.api.ServiceFactoryPrx;
import omero.api.IUpdatePrx;
import omero.model.ImageI;
import omero.model.Image;
public class updates {
    public static void main(String args[]) throws Exception {
        omero.client client = new omero.client(args);
        try {
            Image i = new ImageI();
            i.setName( rstring("name") );
            i.setAcquisitionDate( rtime(System.currentTimeMillis()) );
            ServiceFactoryPrx sf = client.createSession();
            IUpdatePrx u = sf.getUpdateService();
            i = (Image) u.saveAndReturnObject( i );
        } finally {
            client.closeSession();
        }
    }
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/updates.java>

Examples

To tie together some of the topics which we have outlined above, we would like to eventually have several more or less complete application examples which you can use to get started. For the moment, there is just one simpler example `TreeList`, but more will certainly be added. Let us know any ideas you may have.

TreeList

```
# No cpp example
```

```

function projects = AllProjects(query, username)
q = ['select p from Project p join fetch p.datasetLinks dil ',...
    'join fetch dil.child where p.details.owner.omeName = :name'];
p = omero.sys.ParametersI();
p.add('name', omero.rtypes.rstring(username));
projects = query.findAllByQuery(q, p);

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/AllProjects.m>

```
import omero
from omero.rtypes import *
from omero_sys_ParametersI import ParametersI
def getProjects(query_prx, username):
    return query_prx.findAllByQuery(
        "select p from Project p join fetch p.datasetLinks dil join fetch dil.child
↪where p.details.owner.omeName = :name",
        ParametersI().add("name", rstring(username)))
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/AllProjects.py>

```
import java.util.List;
import omero.model.Project;
import omero.api.IQueryPrx;
import omero.sys.ParametersI;
import static omero.rtypes.*;
public class AllProjects {
    public static List<Project> getProjects(IQueryPrx query, String username) throws
↪Exception {
        List rv = query.findAllByQuery(
            "select p from Project p join fetch p.datasetLinks dil join fetch dil.child
↪where p.details.owner.omeName = :name",
            new ParametersI().add("name", rstring(username)));
        return (List<Project>) rv;
    }
}
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/AllProjects.java>

No cpp example

```
function PrintProjects(projects)
if (projects.size()==0)
    return;
end;
for i=0:projects.size()-1,
    project = projects.get(i);
    disp(project.getName().getValue());
    links = project.copyDatasetLinks();
    if (links.size()==0)
        return
    end
    for j=0:links.size()-1,
        pdl = links.get(j);
        dataset = pdl.getChild();
        disp(sprintf(' %s', char(dataset.getName().getValue())));
    end
end
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/PrintProjects.m>

```
def print_(projects):
    for project in projects:
```

(continues on next page)

(continued from previous page)

```

print(project.getName().val)
for pdl in project.copyDatasetLinks():
    dataset = pdl.getChild()
    print(" " + dataset.getName().val)

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/PrintProjects.py>

```

import java.util.List;
import omero.model.Project;
import omero.model.ProjectDatasetLink;
import omero.model.Dataset;
public class PrintProjects {
    public static void print(List<Project> projects) {
        for (Project project : projects) {
            System.out.print(project.getName().getValue());
            for (ProjectDatasetLink pdl : project.copyDatasetLinks()) {
                Dataset dataset = pdl.getChild();
                System.out.println(" " + dataset.getName().getValue());
            }
        }
    }
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/PrintProjects.java>

```

#include <omero/client.h>
#include <Usage.h>
#include <AllProjects.h>
#include <PrintProjects.h>
int main(int argc, char* argv[]) {
    std::string host, port, user, pass;
    try {
        host = argv[0];
        port = argv[1];
        user = argv[2];
        pass = argv[3];
    } catch (...) {
        Usage::usage();
    }
    omero::client client(argc, argv);
    int rc = 0;
    try {
        omero::api::ServiceFactoryPrx factory = client.createSession(user, pass);
        std::vector<omero::model::ProjectPtr> projects =
        ↪AllProjects::getProjects(factory->getQueryService(), user);
        PrintProjects::print(projects);
    } catch (...) {
        client.closeSession();
    }
    return rc;
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/Main.cpp>

```
function Main(varargin)
try
    host = varargin{1};
    port = varargin{2};
    user = varargin{3};
    pass = varargin{4};
catch ME
    Usage
end
client = omero.client(host, port);
factory = client.createSession(user, pass);
projects = AllProjects(factory.getQueryService(), user);
PrintProjects(projects);
client.closeSession();
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/Main.m>

```
import sys
import omero
import Usage, AllProjects, PrintProjects
if __name__ == "__main__":
    try:
        host = sys.argv[1]
        port = sys.argv[2]
        user = sys.argv[3]
        pasw = sys.argv[4]
    except:
        Usage.usage()
    client = omero.client(sys.argv)
    try:
        factory = client.createSession(user, pasw)
        projects = AllProjects.getProjects(factory.getQueryService(), user)
        PrintProjects.print_(projects)
    finally:
        client.closeSession()
```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/Main.py>

```
import omero.api.ServiceFactoryPrx;
import omero.model.Project;
import java.util.List;
public class Main {
    public static void main(String args[]) throws Exception{
        String host = null, port = null, user = null, pass = null;
        try {
            host = args[0];
            port = args[1];
            user = args[2];
            pass = args[3];
        } catch (Exception e) {
            Usage.usage();
        }
        omero.client client = new omero.client(args);
```

(continues on next page)

(continued from previous page)

```

    try {
        ServiceFactoryPrx factory = client.createSession(user, pass);
        List<Project> projects = AllProjects.getProject(factory.getQueryService(),
↪user);
        PrintProjects.print(projects);
    } finally {
        client.closeSession();
    }
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/TreeList/Main.java>

Advanced topics

Sudo

If you are familiar with the admin user concept in OMERO, you might wonder if it is possible for administrative users to perform tasks for regular users. Under Unix-based systems this is commonly known as “sudo” functionality. Although not (yet) as straightforward, it is possible to create sessions for other users and carry out actions on their behalf.

```

#include <iostream>
#include <omero/api/IAdmin.h>
#include <omero/api/ISession.h>
#include <omero/client.h>
#include <omero/model/Session.h>
int main(int argc, char* argv[]) {
    Ice::StringSeq args1 = Ice::argsToStringSeq(argc, argv);
    Ice::StringSeq args2(args1); // Copies
    // ticket:1246
    Ice::InitializationData id1;
    id1.properties = Ice::createProperties(args1);
    Ice::InitializationData id2;
    id2.properties = Ice::createProperties(args2);
    omero::client_ptr client = new omero::client(id1);
    omero::client_ptr sudoClient = new omero::client(id2);
    omero::api::ServiceFactoryPrx sf = client->createSession();
    omero::api::ISessionPrx sessionSvc = sf->getSessionService();
    omero::sys::PrincipalPtr p = new omero::sys::Principal();
    p->name = "root"; // Can change to any user
    p->group = "user";
    p->eventType = "User";
    omero::model::SessionPtr sudoSession = sessionSvc->createSessionWithTimeout( p,
↪3*60*1000L ); // 3 minutes to live
    omero::api::ServiceFactoryPrx sudoSf = sudoClient->joinSession( sudoSession->
↪getUuid()->getValue() );
    omero::api::IAdminPrx sudoAdminSvc = sudoSf->getAdminService();
    std::cout << sudoAdminSvc->getEventContext()->userName;
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/sudo.cpp>

```

client = omero.client();
sudoClient = omero.client();
try
    sf = client.createSession('root','ome');
    sessionSvc = sf.getSessionService();
    p = omero.sys.Principal();
    p.name = 'root'; % Can change to any user
    p.group = 'user';
    p.eventType = 'User';
    sudoSession = sessionSvc.createSessionWithTimeout( p, 3*60*1000 ); % 3 minutes to
↪live
    sudoSf = sudoClient.joinSession( sudoSession.getUuid().getValue() );
    sudoAdminSvc = sudoSf.getAdminService();
    disp(sudoAdminSvc.getEventContext().userName);
catch ME
    sudoClient.closeSession();
    client.closeSession();
end

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/sudo.m>

```

import sys
import omero
args = list(sys.argv)
client = omero.client(args)
sudoClient = omero.client(args)
try:
    sf = client.createSession("root", "ome")
    sessionSvc = sf.getSessionService()
    p = omero.sys.Principal()
    p.name = "root" # Can change to any user
    p.group = "user"
    p.eventType = "User"
    sudoSession = sessionSvc.createSessionWithTimeout( p, 3*60*1000L ) # 3 minutes to
↪live
    sudoSf = sudoClient.joinSession( sudoSession.getUuid().getValue() )
    sudoAdminSvc = sudoSf.getAdminService()
    print(sudoAdminSvc.getEventContext().userName)
finally:
    sudoClient.closeSession()
    client.closeSession()

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/sudo.py>

```

import java.util.List;
import omero.api.IAdminPrx;
import omero.api.ISessionPrx;
import omero.api.ServiceFactoryPrx;
import omero.model.Session;
import omero.sys.Principal;
public class sudo {
    public static void main(String args[]) throws Exception {
        omero.client client = new omero.client(args);

```

(continues on next page)

(continued from previous page)

```

omero.client sudoClient = new omero.client(args);
try {
    ServiceFactoryPrx sf = client.createSession("root", "ome");
    ISessionPrx sessionSvc = sf.getSessionService();
    Principal p = new Principal();
    p.name = "root"; // Can change to any user
    p.group = "user";
    p.eventType = "User";
    Session sudoSession = sessionSvc.createSessionWithTimeout( p, 3*60*1000L ); /
↪ / 3 minutes to live
    ServiceFactoryPrx sudoSf = sudoClient.joinSession( sudoSession.getUuid().
↪ getValue() );
    IAdminPrx sudoAdminSvc = sudoSf.getAdminService();
    System.out.println( sudoAdminSvc.getEventContext().userName );
} finally {
    sudoClient.closeSession();
    client.closeSession();
}
}
}

```

Example: <https://github.com/ome/openmicroscopy/blob/develop/examples/OmeroClients/sudo.java>

Proposed

Like the complete examples above, there are several topics which need to be covered in more detail:

- how to detect client/server version mismatches
- how to make asynchronous methods
- how to use client callbacks
- how to make use of your own ObjectFactory

Planned improvements and known issues

Topics to be added

Obviously, this introduction is still not exhaustive by any means. Some topics which we would like to see added here in the near future include:

- more examples of working with the *OME-Remote Objects*
- examples of all services
- security and ownership
- performance

Code generation

Although not directly relevant to writing a client, it is important to note that much of the code for *OMERO Python language bindings*, *OMERO C++ language bindings*, and *OMERO Java language bindings* is code generated by the BlitzBuild. Therefore, many of the imported and included files in the examples above cannot be found in [github](#).

We plan to include packages of the generated source code in future releases. Until then, it is possible to find our latest builds on [jenkins](#) or to build them locally, although some of the generated files are later overwritten by hand-written versions:

- model is located in `components/tools/OmeroCpp/src/omero/model/`
- OmeroPy is located in `components/tools/OmeroPy/src/`

Lazy loading and caching

Separate method calls will often return one and the same object, say `Dataset#123`. Your application, however, will not necessarily recognize them as the same entity unless you explicitly check the id value. A client-side caching mechanism would allow duplicate objects to be handled transparently, and would eventually facilitate lazy loading.

Helper classes

Several types are harder to use than they need be. `omero.sys.Parameters`, for example, is a class for which native implementations are quite helpful. We have provided `omero.sys.ParametersI` in all supported languages, and will most likely support more over time:

Other

- Superclasses need to be introduced where possible to replace the `ome.model.*` interfaces
- Annotation-link-loading can behave strangely if `AnnotationLink.child` is not loaded.

3.7.2 OMERO Application Programming Interface

All interaction with the OMERO server takes place via several API services available from a ServiceFactory. A service factory is obtained from the client connection e.g. Python:

```
import omero.clients

client = omero.client("localhost")
session = client.createSession("username", "password")  # this is the service factory
adminService = session.getAdminService()                # now we can get/create services
```

- The *Service factory API* has methods for creating Stateless and Stateful services, see below.
 - Stateless services are obtained using “getXXX” methods e.g. `getQueryService()`
 - Stateful services are obtained using “createXXX” methods e.g. `createRenderingEngine()`
- Services will provide access to `omero.model.objects`. You will then need the *API for these objects*, e.g. `Dataset`, `Image`, `Pixels`, etc.

- Some services or their operations may be marked as being deprecated. You may use them but do seek [developer support](#) if you rely on them and can find no alternative as the deprecation means that *you are at risk of our removing them with no further notice*.

Services list

The `ome.api` package in the common component defines the central “verbs” of the OMERO system. All external interactions with the system should happen with these verbs, or services. Each OMERO service belongs to a particular service level with each level calling only on services from lower levels.

Service Level 1 (direct database and Hibernate connections)

- AdminService: [src](#), [API](#) for working with Experimenters, Groups and the current Context (switching groups etc.).
- ConfigService: [src](#), [API](#) for getting and setting config parameters.
- ContainerService: [API](#) for loading Project, Dataset, Image, Screen, Plate hierarchies.
- LdapService: [src](#), [API](#) for communicating with LDAP servers.
- MetadataService: [API](#) for working with Annotation and retrieving acquisition metadata e.g. instrument.
- PixelsService: [API](#) for pixels stats and creating Images with existing or new Pixels.
- ProjectionService [API](#) for performing projections of Pixels sets.
- QueryService: [src](#), [API](#) for custom SQL-like queries.
- RenderingSettingsService [API](#) for copying, pasting & resetting rendering settings.
- RepositoryInfo [API](#) disk space stats.
- RoiService [API](#) working with ROIs (now deprecated).
- ScriptService [API](#) for uploading and launching Python scripts.
- SessionService [API](#) for creating and working with OMERO sessions.
- ShareService [API](#) (now deprecated).
- TimelineService [API](#) for queries based on time.
- TypesService [API](#) for Enumerations.
- UpdateService: [src](#), [API](#) for saving and editing `omero.model` objects.

Service Level 2

- [IContainer](#)
- [ITypes](#)

Stateful/Binary Services

- RawFileStore: [src](#), [API](#) for reading and writing files
- RawPixelsStore: [src](#), [API](#) for reading and writing pixels data
- RenderingEngine: [src](#), [API](#) for viewing images, see *OMERO rendering engine* for more details
- ThumbnailStore: [src](#), [API](#) for retrieving thumbnails
- [IScale](#) for scaling rendered images

A complete list of service APIs can be found [here](#) and some examples of API usage in Python are provided. Java or C++ code can use the same API in a very similar manner.

Discussion

Reads and writes

IQuery and IUpdate are the basic building blocks for the rest of the (non-binary) API. IQuery is based on QuerySources and QueryParameters which are explained under *Using server queries internally*. The goal of this design is to make wildly separate definitions of queries (templates, db-stored, Java code, C# code, ...) runnable on the server.

IUpdate takes any graph composed of IObject objects and checks them for dirtiness. All changes to the graph are stored in the database if the user calling IUpdate has the proper permissions, otherwise an exception is thrown.

Dirty checks follow the Three Commandments:

1. Any IObject-valued field with unloaded set to true is treated as a proxy and is reloaded from the database.
2. Any collection-valued field with a null value is re-loaded from the database.
3. Any collection-valued field with the FILTERED flag is assumed to be dirty and is loaded from the database, with the future option of examining the filtered collection for any new and updated values and applying them to the real collection. Deletions cannot happen this way since it would be unclear if the object was filtered or deleted.

Administration

The IAdmin interface defines all the actions necessary to administer the *Server security and firewalls*. It is explained further on the *OMERO admin interface* page.

Model Object Java

Certain operations, like those dealing with data management and viewing, happen more frequently than others e.g. defining microscopes. Those have been collected in the IContainer interface. IContainer simplifies a few very common queries, and there is a related package `omero.gateway.model.*` for working with the returned graphs. OMERO.insight works almost exclusively with the IContainer interface mostly indirectly via the *Java Gateway*.

Examples

```
// Saving a simple change
Dataset d = iQuery.get(Dataset.class, 1L);
d.setName("test");
iUpdate.saveObject(d);

// Creating a new object
Dataset d = new Dataset();
d.setName("test"); // not-null fields must be filled in
iUpdate.saveObject(d);

// Retrieving a graph
Set<Dataset> ds = iQuery.findAllByQuery("from Dataset d left outer join d.images where d.
↪name = 'test'", null);
```

Stateless versus stateful services

A stateless service has no client-noticeable lifecycle and all instances can be treated equally. A new stateful service, on the other hand, will be created for each client-side proxy, see the `ServiceFactory.createXXX` methods. Once obtained, a stateful service proxy can only be used by a single user. After task completion, the service should be closed i.e. `proxy.close()` to free up server resources.

How to write a service

A tutorial is available at [How To create a service](#). See *Build System* for more information on how the annotated service will be deployed. In the case of *OMERO.blitz*, the service must be properly defined under <https://github.com/ome/omero-blitz/tree/v5.7.2/src/main/slice/omero>.

OMERO annotations for validation

The server-side implementation of these interfaces makes use of [Java annotations](#) and an *AOP* interceptor to validate all method parameters. Calls to `pojos.findContainerHierarchies` are first caught by a method interceptor, which checks for annotations on the parameters and, if available, performs the necessary checks. The interceptor also makes proactive checks. For a range of parameter types such as Java Collections it requires that annotations exist and will refuse to proceed if not implemented.

An API call of the form:

```
pojos.findContainerHierarchies(Class, Set, Map)
```

is implemented as

```
pojos.findContainerHierarchies(@NotNull Class, @NotNull @Validate(Integer.class) Set, ↪
↪Map)
```

See also:

Using server queries internally, OMERO rendering engine, Exception handling

3.7.3 OMERO admin interface

The one central interface for administering the OMERO security system is `IAdmin`. Though several of the methods are restricted to system users (root and other administrators), many are also for general use. The `RolesAllowed` annotations on the `LocalAdmin` class define who can use which methods.

Actions available through `IAdmin` and `IUpdate`

A couple of the methods in the `IAdmin` interface are also available implicitly through `IUpdate`, the main interface for updating the database. This duplication is mainly useful for large scale changes, such as changing the permissions to an entire object graph.

- `changePermissions`
- `changeGroup`

The following shows how these methods can be equivalently used:

```
// setup
ServiceFactory sf = new ServiceFactory();
IAdmin iAdmin = sf.getAdminService();
IUpdate iUpdate = sf.getUpdateService();
Image myImg = ... ; //

// using IAdmin -- let's change the group of myImg
// and then make it group private.
iAdmin.changeGroup(myImg, new ExperimenterGroup( 3L, false ));
iAdmin.changePermissions( myImg, new Permissions( Permissions.GROUP_PRIVATE ));

// and do the same using Details and IUpdate
myImg.getDetails().setPermissions( new Permissions( Permissions.GROUP_PRIVATE ));
myImg.getDetails().setGroup( new ExperimenterGroup( 3L, false ));
iUpdate.saveObject( myImg );
```

The benefit of the second method is the batching of changes into a single call. The benefit of the first is at most explicitness. Note, however, that changing any of the values of `Details` which are not also changeable through `IAdmin` will result in a `SecurityViolation`.

Actions only available through `IAdmin`

The rest of the write methods provided by `IAdmin` are disallowed for `IUpdate` and will throw `SecurityViolations`. This includes adding users, groups, user/group maps, events, enums, or similar. (Enums here are a special case, because they are created not through `IAdmin` but through `ITypes`). A system administrator may be able to use `IUpdate` to create these “System-Types” but using `IAdmin` is safer, cleaner, and guaranteed to work in the future.

The password methods and `synchronizeLoginCache` are also special cases in that they have no equivalent in any other API.

Similarities between IAdmin and IQuery

All of the read methods provided by IAdmin are also available from IQuery, that is, the IAdmin (currently) provide no special context or security privileges. However, having all of the methods in one interface reduces code duplication, which is especially useful when you want the entire user/group graph as provided by `getExperimenter/getGroup/lookupExperimenter/lookupGroup`.

See also:

OMERO Application Programming Interface

3.7.4 Deleting in OMERO

Deleting data in OMERO is complex due to the highly linked nature of data in the database. For example, an Image has links to Datasets, Comments, Tags, Instrument, Acquisition metadata etc. If the image is deleted, some of this other data should remain and some should be deleted with the image (since it has no other relevance).

In the 4.2.1 release of OMERO, an improved deleting service was introduced to fix several problems or requirements related to the delete functionality (see #2615 for tickets):

- Need a better way to define what gets deleted when certain data gets deleted (e.g. Image case above)
- Need to be able to configure this definition, since different users have different needs
- Deleting large amounts of data (e.g. Plate of HCS data) was too memory-intensive (data was loaded from the database during delete)
- Poor logging of deletes
- Large deletes (e.g. screen data) take time: Clients need to be able to keep working while deletes run ‘in the background’
- Binary data (pixels, thumbnails, files etc) was not removed at delete time - required sysadmin to clean up later

Future releases will continue this work (see #2911) and the 5.1.0 release of OMERO offers a new implementation of deletion.

Finality of deletion

Import in OMERO 5.x uploads the image and companion files intact and stores them within subdirectories of the directory configured by the value of `omero.managed.dir`, typically `ManagedRepository`. The files relating to a specific fileset are stored together on the server’s filesystem and they are read by Bio-Formats when images are viewed in OMERO clients. If any of a fileset’s files, or the corresponding entries for them in the database, are deleted, then the fileset may no longer be readable. If all the fileset’s files are deleted, the fileset will certainly be unreadable, and there is no ‘undo’ that will bring it back.

Delete behavior (technical)

Configuring what gets deleted is done using XML files. Since OMERO 5.1, the delete behavior defaults to a *Model graph operations* implementation that is configured by <https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/resources/ome/services/blitz-graph-rules.xml>.

Delete Image

The general delete behavior for deleting an Image is to remove every piece of data from the database that was added when the image was imported, removing pixel data and thumbnails from disk. In addition, the following data is deleted:

- Comments on the image
- Rating of the image
- ROIs for this image (see below)
- Image Rendering settings for yourself and other users

Optional - In OMERO.web and OMERO.insight, you will be asked whether you also want to delete:

- Files attached to the image (if not linked elsewhere). In that case, the binary data will be removed from disk too.
- Your own tags on the image (if not used elsewhere)

The same option is available when deleting dataset, project, plate, screen.

Delete Dataset or Project

When deleting a Project or Dataset, you have the option to also delete tags and annotations (as for Image above). You also can choose whether to 'delete contents'. This will delete any Datasets (or Images) that are contained in the Project (or Dataset). However, Datasets and Images will not get deleted if they are also contained in other Projects or Datasets respectively.

If a user decides to delete/keep the annotations (see **Optional** above) when deleting a Project (or Dataset) and its contents, the rule associated to the annotation will be apply to all objects.

Delete Screen, Plate or Plate Acquisition

When deleting a Screen, you have the option to also delete tags and annotations. You also can choose whether to 'delete contents'. This will delete any Plates that are contained in the Screen. However, Plates will not get deleted if they are also contained in other Screen.

When deleting a Plate, you have the option to also delete tags and annotations but **NOT** the option to 'delete contents'.

If the Plate has Plate Acquisitions, you can delete one or more Plate Acquisition at once.

Delete Tag/Attachment

You can delete a Tag/Attachment, and it will be removed from all images. However you cannot delete a Tag/Attachment if it has been used by another user in the same collaborative group. This is to prevent potential loss of significant amount of annotation effort by other users. You will need to get the other users to first remove your Tag/Attachment where they have used it, before you can delete it.

Known Issue: if the owner of the Tag/Attachment is also an owner of the group (e.g. PI), they will be able to delete their Tag/Attachment, even if others have used it.

Delete multi-file Images and Image sets

An Image, or a set of Images, may come from a single file or a set of dependent files. For instance, a single Leica LIF file may contain many Images, as may a Zeiss mdb file with lsm files. On the other hand, some file formats, like Deltavision with log file, or the original ICS file format, use multiple files to represent a single Image. At import time, these groups of related files and Images are organized into Filesets: a Fileset is a set of files that encode a set of Images. The simplest case where there is one file per Image still has a corresponding Fileset.

Even if many Images come from the same file, they may be separately selected and viewed in client software. However, at least at present, a Fileset may not be partially deleted: either all the files and Images from it are deleted, or none are. So, for instance, the Images from the same Leica LIF file may be deleted only all at once, and the Deltavision log file is not deleted separately from the main file. The same applies to high-content screening data: a Plate with its Wells and Images are all stored in one Fileset and may be deleted only together.

Each Fileset has a corresponding directory on the server in which, perhaps in subdirectories, all its files are stored. All the file paths for an Image's Fileset can be accessed from the tool-bar at the top of the right-hand panel.

Delete in collaborative group

Some more discussion of delete issues in a collaborative group, where your data are linked to data of other users, can be found on the [Groups and permissions system](#) page.

- A user cannot remove Images from another user's Dataset, or remove Datasets (or Plates) from Projects (or Screens).
- A user cannot delete anything that belongs to another user.

Group owner rights

An owner of the group, usually a PI, can delete anything that belongs to other members of the group.

Edge cases

These are 'known issues' that may cause problems for some users (not for most). These will be resolved in future depending on priority.

- Other users' ROIs (and associated measurements) are deleted from images.
- Multiply-linked objects are unlinked and not deleted e.g. Project p1 contains two Datasets d1 and d2, Project p2 contains Dataset d1. If the Project p1 is deleted, the Dataset d1 is only unlinked from p1 and not completely deleted.

Binary data

When Images, Plates or File Annotations have been successfully deleted from the database the corresponding binary data is deleted from the [binary repository](#). It is possible that some files may not be successfully deleted if they are locked for any reason. In this case, the undeleted files can be removed manually via **omero admin cleanse**. This also deletes any empty directories left behind after the binary data that they contained has been deleted.

3.7.5 OMERO Import Library

The Import Library is a re-usable framework for building import clients. Several are provided by the OMERO team directly:

- the integrated *importer*
- *Command Line Importer* tool

Components

The primary classes which make up the Import Library are:

- `ImportLibrary.java` itself, which is the main driver
- `ImportCandidates.java` which takes file paths and determines the proper files to import
- `ImportConfig.java`, an extensible mechanism for storing the properties used during import
- `ImportEvent.java`, the various events raised during import to `IObserver` and `IObservable` implementations
- `OMEROMetadataStoreClient.java`, the low-level connection to the server
- `OMEROWrapper.java`, the OMERO adapter for the Bio-Formats `ImageReaders` class
- In OMERO.insight, the main entry point is the `importImage` method of `OMEROGateway.java`
- In the CLI, the main entry point is the `CommandLineImporter` class

Earlier Import Workflow

Prior to OMERO 5.0, the import workflow was very much client-side. Using the `ImportLibrary` a client would determine the import candidates and then import the image. The import phase would comprise copying the pixel data to the OMERO data directory, writing the metadata into the database, and optionally copying the original file to the OMERO data directory for archiving.

FS Managed Repository Import Workflow

From 5.0 the workflow has changed. The client still determines the import candidates but the client-side import process simply uploads the original files to the OMERO data directory and then uses the `ManagedRepository` service to initiate a server-side import. On the server the import is then completed by writing the metadata into the database. After import, pixel data is accessed directly from the original files using Bio-Formats. This means that data files are no longer duplicated and any nested directory structure is preserved. It also allows OMERO to take advantage of pre-generated pyramids available in some formats e.g. dedicated whole slide imaging formats such as `SVS` (instead of generating `OMERO` pyramids).

For full details of the import workflow see *Import under OMERO.fs*.

Example

The `CommandLineImporter.java` class shows a straightforward import. An `ErrorHandler` instance is passed both to the `ImportCandidates` constructor (since errors can occur while parsing a directory) and to the `ImportLibrary`. This and other handlers receive `ImportEvents` which notify listeners of the state of the current import.

3.7.6 TempFileManager

Class to be used by *Working with OMERO* and server components to allow a uniform creation of temporary files and folders with a best-effort guarantee of deleting the resources on exit. The manager searches three locations in order, taking the first which allows lockable write-access (See #1653):

- The environment property setting `OMERO_TMPDIR`
- The user's home directory, for example specified in Java via `System.getProperty("user.home")`
- The system temp directory, in Java `System.getProperty("java.io.tmpdir")` and in Python `tempfile.gettempdir()`

Creating temporary files

For the user “ralph”,

```
from omero.util.temp_files import create_path
path = create_path("omero", ".tmp")
```

or

```
import omero.util.TempFileManager
File file = TempFileManager.create_path("omero", ".tmp")
```

both produce a file under the directory:

```
/tmp/omero_ralph/$PID/omero$RANDOM.tmp
```

where `$PID` is the current process id and `$RANDOM` is some random sequence of alphanumeric characters.

Removing files

If `remove_path` is called on the return value of `create_path`, then the temporary resources will be cleaned up immediately. Otherwise, when the Java or Python process exits, they will be deleted. This is achieved in Java through `Runtime#addShutdownHook(Thread)` and in Python via `atexit.register()`.

Creating directories

If an entire directory with a unique directory is needed, pass “true” as the “folder” argument of the `create_path` method:

```
create_path("omero", ".tmp", folder = True)
```

and

```
TempFileManager.create_path("omero", ".tmp", true);
```

Note: All contents of the generated directory will be deleted.

See also:

#1534

3.7.7 Exception handling

Client exceptions

The exceptions which can be received by a client due to a remote call on the OMERO server are all defined in <https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/slice/omero/ServerError.ice> (included below). This file contains two separate hierarchies rooted at `Ice::Exception` and `omero::ServerError`.

For a better understanding of how to handle exceptions, please read both of the *.ice files carefully, and see [Working with OMERO](#) for examples of exception handling.

```
/*
 *   $Id$
 *
 *   Copyright 2007 Glencoe Software, Inc. All rights reserved.
 *   Use is subject to license terms supplied in LICENSE.txt
 *
 */
#ifndef OMERO_SERVERERRORS_ICE
#define OMERO_SERVERERRORS_ICE
#include <Glacier2/Session.ice>
/**
 * Exceptions thrown by OMERO server components. Exceptions thrown client side
 * are available defined in each language binding separately, but will usually
 * subclass from "ClientError"
 *
 * including examples of what a appropriate try/catch block would look like.
 *
 * <p>
 * All exceptions that are thrown by a remote call (any call on a *Prx instance)
 * will be either a subclass of [Ice::UserException] or [Ice::LocalException].
 * <a href="https://doc.zeroc.com/display/Ice/Run-Time+Exceptions#Run-TimeExceptions-
 * InheritanceHierarchyforExceptions">Inheritance Hierarchy for Exceptions</a>
 * from the Ice manual shows the entire exception hierarchy. The exceptions described in
 * this file will subclass from [Ice::UserException]. Other Ice-runtime exceptions
 * subclass
 * from [Ice::LocalException].
 * </p>
 *
 * <pre>
 * OMERO Specific:
 * =====
```

(continues on next page)

(continued from previous page)

```

*   ServerError (root server exception)
*   |
*   |_ InternalException (server bug)
*   |
*   |_ ResourceError (non-recoverable)
*   |   \_ NoProcessorAvailable
*   |
*   |_ ConcurrencyException (recoverable)
*   |   |_ ConcurrentModification (data was changed)
*   |   |_ OptimisticLockException (changed data conflicts)
*   |   |_ LockTimeout (took too long to acquire lock)
*   |   |_ TryAgain (some processing required before server is ready)
*   |   \_ TooManyUsersException
*   |       \_ DatabaseBusyException
*   |
*   |_ ApiUsageException (misuse of services)
*   |   |_ OverUsageException (too much)
*   |   |_ QueryException (bad query string)
*   |   \_ ValidationException (bad data)
*   |
*   |_ SecurityViolation (some no-no)
*   |   \_ GroupSecurityViolation
*   |       |_ PermissionMismatchGroupSecurityViolation
*   |       \_ ReadOnlyGroupSecurityViolation
*   |
*   \_ SessionException
*       |_ RemovedSessionException (accessing a non-existent session)
*       |_ SessionTimeoutException (session timed out; not yet removed)
*       \_ ShutdownInProgress (session on this server will most likely be destroyed)
* </pre>

```

However, in addition to [Ice::LocalException] subclasses, the Ice runtime also defines subclasses of [Ice::UserException]. In some cases, OMERO subclasses from these exceptions. The subclasses shown below are not exhaustive, but show those which an application's exception handler may want to deal with.

```

* <pre>
*   Ice::Exception (root of all Ice exceptions)
*   |
*   |_ Ice::UserException (super class of all application exceptions)
*   |   |
*   |   |_ Glacier2::CannotCreateSessionException (1 of 2 exceptions throwable by
*   |   |   ↪ createSession)
*   |   |   |_ omero::AuthenticationException (bad login)
*   |   |   |_ omero::ExpiredCredentialException (old password)
*   |   |   |_ omero::WrappedCreateSessionException (any other server error during
*   |   |   ↪ createSession)
*   |   |   \_ omero::licenses::NoAvailableLicensesException (see tools/licenses/

```

(continues on next page)

(continued from previous page)

```

↪resources/omero/LicensesAPI.ice)
*   | |
*   | \_ Glacier2::PermissionDeniedException (other of 2 exceptions throwable by ↪
↪createSession)
*   |
*   \_ Ice::LocalException (should generally be considered fatal. See exceptions below)
*       |
*       |_ Ice::ProtocolException (something went wrong on the wire. Wrong version?)
*       |
*       |_ Ice::RequestFailedException
*       |   |_ ObjectNotExistException (Service timeout or similar?)
*       |   \_ OperationNotExistException (Improper use of uncheckedCast?)
*       |
*       |_ Ice::UnknownException (server threw an unexpected exception. Bug!)
*       |
*       \_ Ice::TimeoutException
*           \_ Ice::ConnectTimeoutException (Couldn't establish a connection. Retry?)
*
* </pre>
**/
module omero
{
    /*
    * Base exception. Equivalent to the ome.conditions.RootException.
    * RootException must be split into a ServerError and a ClientError
    * base-class since the two systems are more strictly split by the
    * Ice-runtime than is done in RMI/Java.
    */
    exception ServerError
    {
        string serverStackTrace;
        string serverExceptionClass;
        string message;
    };
    // SESSION EXCEPTIONS -----
    /**
    * Base session exception, though in the OMERO.blitz
    * implementation, all exceptions thrown by the Glacier2
    * must subclass CannotCreateSessionException. See below.
    */
    exception SessionException extends ServerError
    {
    };
    /**
    * Session has been removed. Either it was closed, or it
    * timed out and one "SessionTimeoutException" has already
    * been thrown.
    */
    exception RemovedSessionException extends SessionException
    {
    };

```

(continues on next page)

(continued from previous page)

```

/**
 * Session has timed out and will be removed.
 */
exception SessionTimeoutException extends SessionException
{
};
/**
 * Server is in the progress of shutting down which will
 * typically lead to the current session being closed.
 */
exception ShutdownInProgress extends SessionException
{
};
// SESSION EXCEPTIONS (Glacier2) -----
/**
 * createSession() is a two-phase process. First, a PermissionsVerifier is
 * called which must return true; then a SessionManager is called to create
 * the session (ServiceFactory). If the PermissionsVerifier returns false,
 * then PermissionDeniedException will be thrown. This, however, cannot be
 * subclassed and so string parsing must be used.
 */
/**
 * Thrown when the information provided omero.createSession() or more
 * specifically Glacier2.RouterPrx.createSession() is incorrect. This
 * does -not- subclass from the omero.ServerError class because the
 * Ice Glacier2::SessionManager interface can only throw CCSEs.
 */
exception AuthenticationException extends Glacier2::CannotCreateSessionException
{
};
/**
 * Thrown when the password for a user has expired. Use: ISession.
↪changeExpiredCredentials()
 * and login as guest. This does -not- subclass from the omero.ServerError class.
↪because the
 * Ice Glacier2::SessionManager interface can only throw CCSEs.
 */
exception ExpiredCredentialException extends Glacier2::CannotCreateSessionException
{
};
/**
 * Thrown when any other server exception causes the session creation to fail.
 * Since working with the static information of Ice exceptions is not as easy
 * as with classes, here we use booleans to represent what has gone wrong.
 */
exception WrappedCreateSessionException extends Glacier2::CannotCreateSessionException
{
    bool    concurrency;
    long    backOff;    /* Only used if ConcurrencyException */
    string  type;       /* Ice static type information */
};
// OTHER SERVER EXCEPTIONS -----

```

(continues on next page)

(continued from previous page)

```

/**
 * Programmer error. Ideally should not be thrown.
 */
exception InternalException extends ServerError
{
};
// RESOURCE
/**
 * Unrecoverable error. The resource being accessed is not available.
 */
exception ResourceError extends ServerError
{
};
/**
 * A script cannot be executed because no matching processor
 * was found.
 */
exception NoProcessorAvailable extends ResourceError
{
    /**
     * Number of processors that responded to the inquiry.
     * If 1 or more, then the given script was not acceptable
     * (e.g. non-official) and a specialized processor may need
     * to be started.
     */
    int processorCount;
};
// CONCURRENCY
/**
 * Recoverable error caused by simultaneous access of some form.
 */
exception ConcurrencyException extends ServerError
{
    long backOff; /* Backoff in milliseconds */
};
/**
 * Currently unused.
 */
exception ConcurrentModification extends ConcurrencyException
{
};
/**
 * Too many simultaneous database users. This implies that a
 * connection to the database could not be acquired, no data
 * was saved or modified. Clients may want to wait the given
 * backOff period, and retry.
 */
exception DatabaseBusyException extends ConcurrencyException
{
};
/**
 * Conflicting changes to the same piece of data.

```

(continues on next page)

(continued from previous page)

```

    */
exception OptimisticLockException extends ConcurrencyException
{
};
/**
 * Lock cannot be acquired and has timed out.
 */
exception LockTimeout extends ConcurrencyException
{
    int seconds; /* Informational field on how long timeout was */
};
/**
 * Background processing needed before server is ready
 */
exception TryAgain extends ConcurrencyException
{
};
exception MissingPyramidException extends ConcurrencyException
{
    long pixelsID;
};
// API USAGE
exception ApiUsageException extends ServerError
{
};
exception OverUsageException extends ApiUsageException
{
};
/**
 *
 */
exception QueryException extends ApiUsageException
{
};
exception ValidationException extends ApiUsageException
{
};
// SECURITY
exception SecurityViolation extends ServerError
{
};
exception GroupSecurityViolation extends SecurityViolation
{
};
exception PermissionMismatchGroupSecurityViolation extends SecurityViolation
{
};
exception ReadOnlyGroupSecurityViolation extends SecurityViolation
{
};
// OMEROFS
/**

```

(continues on next page)

(continued from previous page)

```

* OmeroFSError
*
* Just one catch-all UserException for the present. It could be
* subclassed to provide a finer grained level if necessary.
*
* It should be fitted into or subsumed within the above hierarchy
**/
exception OmeroFSError extends ServerError
{
    string reason;
};
};
#endif // OMERO_SERVERERRORS_ICE

```

Server exceptions

Due to the strict API boundary enforced by Ice, the client and server exception hierarchies, though related, are distinct. The discussion below is possibly of interest for server developers only. Client developers should refer to the information and examples under *Working with OMERO*.

Interceptor

Exception handling in the OMERO is centralized in an *Aspect-oriented programming* interceptor ([source code](#)). All exceptions thrown by code are caught in a `try {} catch (Throwable t) {}` block. Exceptions which do not subclass `ome.conditions.RootException` are wrapped in an `ome.conditions.InternalException`.

The only exceptions to this are any interceptors which may be run before the exception handler is run. The order of interceptors is defined in [services.xml](#).

Hierarchy

The current exception hierarchy (package `ome.conditions`) used is as follows:

- RootException
 - InternalException - should not reach the client; Bug! Contact administrator e.g. NullPointerException, assertion failed, etc.
 - ResourceError - fatal error in server, e.g. OutOfMemory, disk space full, the database is in illegal state, etc.
 - DataAccessException
 - * SecurityViolation - do not do that! E.g. edit locked project, create new user.
 - * OptimisticLockException - re-load and compare e.g. “someone else has already updated this project”
 - * ApiUsageException - something wrong with how you did things e.g. IllegalStateException, object uninitialized, etc.
 - * ValidationException - something wrong with what you sent; sends list of fields, etc.; edit and retry, e.g. no “?” in image names.

where the colors indicate:

Abstract

FixAndRetryConditions

RetryConditions

NoRecourseConditions

Any other exception which reaches the client should be considered an `OutOfServiceException`, meaning that something is (hopefully only) temporarily wrong with the server, e.g. no connection, server down, server restarting. But since this cannot be caught since the server cannot be reached, there is no way to guarantee that a real `OutOfServiceException` is thrown.

Moving forward

`FixAndRetryConditions` need to have information about what should be fixed, like a `Validation` object which lists fields with error messages. A `RetryCondition` could have a back-off value to prevent too frequent retries.

Questions

- What data should be available in the exceptions?
- What other logic do we want on our exceptions, keeping in mind they will have to be re-implemented in all target languages?

3.7.8 Omero logging

All OMERO components written in Java use the [SLF4J](#) logging facade, typically backed by [Logback](#); all components written in python use the built-in logging module.

Warning: Refrain from calling `logging.basicConfig()` anywhere in your module except in `if __name__ == "__main__":` blocks.

Java clients

Java clients log to `$HOME/omero/log`. The number of files and their size are limited.

`logback-cli.xml` controls the output for the command line importer: all logging goes to standard err, while useful output (pixel ids, or used files) goes to standard out. It is contained within the `omero-blitz.jar` itself. Therefore, to modify the settings use `-Dlogback.configurationFile=/path/to/logback.xml` or similar.

OMERO.insight logging is configured via `logback.xml` which is available in the `config/` directory of any OMERO.insight install.

Java servers

Java server components are configured by passing `-Dlogback.configurationFile=etc/logback.xml` to each Java process. `logback.xml` includes the `scan` attribute so that changes to the logging configuration are [automatically reloaded at regular intervals](#).

By default, the output from logback is sent to: `var/log/<servername>.log`. Once files reach a size of 500MB, they are rolled over to `<servername>.log.1`, `<servername>.log.2`, etc. Once the files have rolled over, you can safely delete or compress (bzip2, gzip, zip) them. Alternatively, once you are comfortable with the stability of your server, you can either reduce logging or the number and size of the files kept. **Note:** if something goes wrong with your server installation, the log files can be very useful in tracking down issues.

In addition, each import process logs to a file under the managed repository which matches the timestamped fileset directory's name. For example, if an imported fileset is uploaded to `/OMERO/ManagedRepository/userA_1/2013-06/17/12-00-00.000`, then the log file can be found under `/OMERO/ManagedRepository/userA_1/2013-06/17/12-00-00.000.log`.

Python servers

Python servers are configured by a call to `omero.util.configure_server_logging(props)`. The property values are taken from the configuration file passed to the server via `icegridnode`. For example, the config file for `Processor-0` can be found in `var/master/servers/Processor-0/config/config`. These values come from `etc/grid/templates.xml`.

All the “`omero.logging.*`” properties can be overwritten in your `etc/grid/default.xml` file. See the “Profile” properties block for how to configure for your site.

Similar to logback, logging is configured to be written to `var/log/<servername>.log` and to maintain 9 backups of at most 500MB.

stdout and stderr

Though all components try to avoid it, some output will still go to stdout/stderr. On non-Windows systems, all of this output will be sent to the `var/log/master.out` and `var/log/master.err` files.

Windows stdout and stderr

On Windows, the state of stdout and stderr is somewhat different. No information will be written to `master.out`, `master.err`, or similar files. Instead, what logging is produced will go to the Windows Event Viewer, but finding error situations can be considerably more challenging (See [#1449](#) for more information).

3.7.9 Graph requests

Overview

The Blitz API offers several requests that are subclasses of `GraphQuery`. These may be submitted to the server for asynchronous processing of linked graphs of *OMERO model objects*. This section gives a brief overview of the graph requests and their purpose. Follow the links to see more details.

Querying the model object graph

GraphQuery (base class)

The parent of the requests below, it includes a `targetObjects` property that specifies from which model objects to start processing. The `LegalGraphTargets` request can be used to determine which types of model object may be targeted.

DiskUsage2

Report on the disk usage of the target objects and their contents by type, user and group. Includes a `targetClasses` property to allow specifying every visible instance of a type.

FindParents

Find the parents of the target objects, both direct and indirect. `typesOfParents` specifies the types of parents to report. `stopBefore` specifies types of model object to avoid in traversing the linked graph upward: those subgraphs are ignored unless otherwise reachable.

FindChildren

Find the children of the target objects, both direct and indirect. `typesOfChildren` specifies the types of children to report. `stopBefore` specifies types of model object to avoid in traversing the linked graph downward: those subgraphs are ignored unless otherwise reachable.

Changing the model object graph

GraphModify2 (base class)

The parent of the requests below, it includes a `targetObjects` property that specifies from which model objects to start processing. The `LegalGraphTargets` request can be used to determine which types of model object may be targeted.

The `childOptions` property lists how to process the contents of targeted objects.

Because these requests change the data stored by the server, a `dryRun` property is provided that enables attempting to obtain the same response or error without actually making any changes.

ChildOption

By default if a ‘child’ object is contained by a ‘parent’ targeted object then it is processed along with its parent if it is not also contained by another parent object that is not targeted. Use requests’ `childOptions` property to specify that children should be processed or not regardless of other parents.

The `includeType` and `excludeType` properties specify for which types of children to override the behavior. For children that are annotations, the `includeNs` and `excludeNs` properties use the annotation namespace to limit the applicability of the override.

Chgrp2

Change the group ID of the targeted objects and their contents. The objects are moved to the group specified by the `groupId` property.

Chown2

Change the user ID of the targeted objects and their contents. The objects are given to the user specified by the `userId` property.

Chmod2

Change the permissions for the targeted objects which must be groups. The `permissions` property specifies the new group type.

Delete2

Delete the targeted objects and their contents. For original file instances the underlying file in the server’s binary repository may be deleted also.

Duplicate

Duplicate a subgraph from the model object graph, starting from the targeted objects and recursing to their contents. The `typesToDuplicate`, `typesToReference`, `typesToIgnore` properties offer control over where in the graph traversal to stop duplicating and with what in the original graph to link the duplicate subgraph.

SkipHead

Defer processing to start only at specific contents of the targeted objects. The `startFrom` property specifies the types of object to actually target with the processing and the `request` property, which may be any of the other requests from this section, specifies what to do to those objects once identified.

Command-line interface

OMERO's *command-line interface client* includes `chgrp`, `chown`, `delete` plugins that construct the corresponding `Chgrp2`, `Chown2`, `Delete2` requests. Additionally, the `group` plugin offers the `Chmod2` request and the `fs` plugin offers the `DiskUsage2` request.

Request builders for Java

The Java gateway includes the `Requests.java` class which offers Java developers a set of builders that use method-chaining to allow convenient construction of new instances of the above requests.

3.7.10 Rewriting old graph requests

Migration is required

OMERO 5.1.0 introduced a new implementation of *Model graph operations* offered through the API via `Chgrp2`, `Chown2`, `Delete2`, and their superclass `GraphModify2`. OMERO 5.1.2 added `Chmod2`. The corresponding deprecated legacy request operations are *removed* in OMERO 5.3. Client code must be adjusted accordingly.

Target objects

For specifying which model objects to operate on, instead of using one request for each object, use `GraphModify2`'s `targetObjects` which allows specification of multiple model object classes, each with an unordered list of IDs, all in a single request. To specify a type, no longer use `/`-delimited paths, but instead just the class name, e.g. `Image` instead of `/Image`. To achieve a root-anchored subgraph operation use `SkipHead` to wrap your request: for instance, for `/Image/Pixels/RenderingDef`, set the `SkipHead` request's `targetObjects` to the image(s), and set `startFrom` to `RenderingDef`.

Translating options

`GraphModify2` offers `childOptions`, an ordered list of `ChildOption` instances, each of which allows its applicability to annotations to be limited by namespace. Some examples:

- To move a dataset with all its images, removing those images from other datasets where necessary, use `Chgrp2` with a `ChildOption`'s `includeType` set to `Image`.
- To delete a dataset without deleting any images at all from it, use `Delete2` with a `ChildOption`'s `excludeType` set to `Image`.
- To delete annotations except for the tags that are in a specific namespace, use `Delete2` with a `ChildOption`'s `excludeType` set to `TagAnnotation` and `includeNs` set to that namespace.

Examples in Python

Move images

```
chgrps = DoAll()
chgrps.requests = [Chgrp(type="/Image", id=n, grp=5) for n in [1,2,3]]
sf.submit(chgrps)
```

used in OMERO 5.0 should now be written as,

```
chgrp = Chgrp2(targetObjects={'Image': [1,2,3]}, groupId=5)
sf.submit(chgrp)
```

Delete plate, but not annotations

```
keepAnn = {"Annotation": "KEEP"}
delete = Delete(type="/Plate", id=8, options=keepAnn)
sf.submit(delete)
```

used in OMERO 5.0 should now be written as,

```
keepAnn = [ChildOption(excludeType=['Annotation'])]
delete = Delete2(targetObjects={'Plate': [8]}, childOptions=keepAnn)
sf.submit(delete)
```

Delete an image's rendering settings

```
delete = Delete(type="/Image/Pixels/RenderingDef", id=6)
sf.submit(delete)
```

used in OMERO 5.0 should now be written as,

```
anchor = {'Image': [6]}
targets = ['RenderingDef']
delete = SkipHead(targetObjects=anchor, startFrom=targets,
                  request=Delete2())
sf.submit(delete)
```

Java request factory

A utility class `Requests.java` provides convenient instantiation of graph requests. This class allows the requests from the above Python examples to be created by,

```
// move images
Chgrp2 example1 = Requests.chgrp().target("Image").id(1L,2L,3L)
    .toGroup(5L).build();

// delete plate, but not annotations
ChildOption childOption = Requests.option()
```

(continues on next page)

(continued from previous page)

```

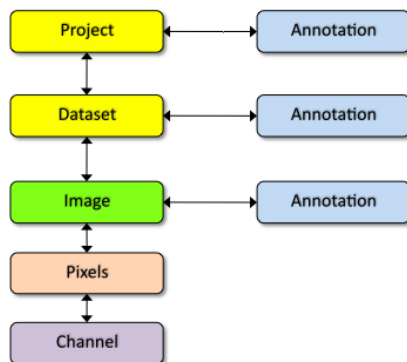
.excludeType("Annotation").build();
Delete2 example2 = Requests.delete().target("Plate").id(8L)
    .option(childOption).build();

// delete an image's rendering settings
SkipHead example3 = Requests.skipHead().target("Image").id(6L)
    .startFrom("RenderingDef").request(Delete2.class).build();

```

3.8 The OME Data Model

3.8.1 OME-Remote Objects



OMERO is based on the OME data model which can appear overly complex for new users. However, the core entities you need for getting started are much simpler.

Images in OMERO are organized into a many-to-many container hierarchy: “Project” -> “Dataset” -> “Image”. These containers (and various other objects) can be annotated to link various types of data. Annotation types include Comment (string), Tag (short string), Boolean, Long, Xml, File attachment etc.

Images are represented as Pixels with 5 dimensions: X, Y, Z, Channel, Time.

At the core of the work on the [Open Microscopy Environment](#) is the definition of a vocabulary for working with microscopic data. This vocabulary has a representation in the [XML specification](#), in the database (the data model), and in code. This last representation is the object model with which we will concern ourselves here.

Because of its complexity, the object model is generated from a [central definition](#) using our own [code-generator](#). It relies on no libraries and can be used in both the server and the RMI clients. The relationships among the objects are enumerated in a cross-referenced [reference document](#). *OMERO.blitz* uses a [second mapping](#) to generate *OMERO Java language bindings*, *OMERO Python language bindings*, and *OMERO C++ language bindings* classes, which can be mapped back and forth to the server object model. *This document discusses only the server object-model and how it is used internally.*

Instances of the object model have no direct interaction with the database, rather the mapping is handled externally by the O/R framework, [Hibernate](#). That means, by and large, generated classes are data objects, composed only of getter and setter fields for fields representing columns in the database, and contain no business logic. However, to make working with the model easier, and perhaps more powerful, there are several features which we have built in.

Note: The discussion here of object types is still relevant but uses the `ome.model.*` objects for examples. These are server internal types which may lead to some confusion. Clients work with `omero.model.*` objects. This documentation will eventually be updated to reflect both hierarchies.

OMERO type language

The Model has two general parts: first, the long-studied and well-established core model and second, the user-specified portion. It is vital that there is a central definition of both parts of the object model. To allow users to easily define new types, we need a simple domain specific language (or little language) which can be mapped to Hibernate mapping files. See an example in <https://github.com/ome/omero-model> at:

- <https://github.com/ome/omero-model/blob/v5.6.13/src/main/resources/mappings/acquisition.ome.xml>

From this DSL, various artifacts can be generated: XML Schema, Java classes, SQL for generating tables, etc. The ultimate goal is to have no exceptions in the model.

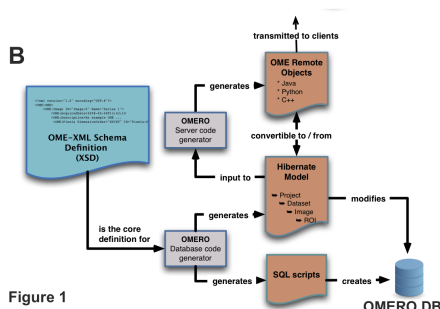


Figure 1

Conceptually, the XSD files under the `components/specification` source directory are the starting point for all code generation. Currently however, the files in <https://github.com/ome/omero-model> under <https://github.com/ome/omero-model/tree/v5.6.13/src/main/resources/mappings> are hand-written based on the XSD files.

The task created from the <https://github.com/ome/omero-dsl-plugin/tree/v5.5.2/src> Java files is then used to turn the mapping

files into generated Java code in <https://github.com/ome/omero-model> under the `build/classes/java/main` directory. These classes are all within the `ome.model` package. A few hand-written Java classes can also be found in <https://github.com/ome/omero-model/tree/v5.6.13/src/main/java/ome/model/internal>.

Warning: The following paragraph is **NOT** up-to-date. Using `build-schema` no longer exists in 5.5.0 and has not been replaced yet.

The `build-schema` ant target takes the generated `ome.model` classes as input and generates the <https://github.com/ome/openmicroscopy/tree/develop/sql/psql> scripts which get used by **omero db script** to generate a working OMERO database. Files named like `OMEROVERSION__PATCH.sql` are hand-written update scripts.

The primary consumer of the `ome.model` classes at runtime is the <https://github.com/ome/omero-server>.

The above classes are considered the internal server code, and are the only objects which can take part in Hibernate transactions.

External to the server code is the <https://github.com/ome/omero-blitz> layer. These classes are in the `omero.model` package. They are generated by another call to the DSL task in order to generate the Java, Python, C++, and Ice files under, by default, `build/psql/`.

In <https://github.com/ome/omero-blitz>, the generated Ice files along with the hand-written Ice files from <https://github.com/ome/omero-blitz/tree/v5.7.2/src/main/slice/omero> are then run through the `slice2cpp`, `slice2java`, and `slice2py` command-line utilities in order to generate source code in each of these languages. Clients pass in instances of these `omero.model` (or in the case of C++, `omero::model`) objects. These are transformed to `ome.model` objects, and then persisted to the database.

If we take a concrete example, a C++ client might create an Image via `new omero::model::ImageI()`. The “I” suffix represents an “implementation” in the Ice naming scheme and this subclasses from `omero::model::Image`. This can be remotely passed to the server which will be deserialized as an `omero.model.ImageI` object. This will then get converted to an `ome.model.core.Image`, which can finally be persisted to the database.

Keywords

Some words are not allowed as properties/fields of OMERO types. These include:

- id
- version
- details
- ... any SQL keyword

Improving generated data objects

Constructors

Two special constructors are generated for each model object. One is for creating proxy instances, and the other is for filling all NOT-NULL fields:

```
Pixels p_proxy = new Pixels(Long, boolean);
Pixels p_filled = new Pixels(ome.model.core.Image, ome.model.enums.PixelType,
    java.lang.Integer, java.lang.Integer, java.lang.Integer, java.lang.Integer, java.
↪lang.Integer,
    java.lang.String, ome.model.enums.DimensionOrder, ome.model.core.
↪PixelsDimensions);
```

The first should almost always be used as: `new Pixels(5L, false)`. Passing in an argument of `true` would imply that this object is actually loaded, and therefore the server would attempt to null all the fields on your object. See below for a discussion on loadedness.

In the special case of Enumerations, a constructor is generated which takes the value field for the enumeration:

```
Format file_format = new Format("text/plain");
```

Further, this is the only example of a managed object which will be loaded by the server **without** its id. This allows applications to record only the string "text/plain" and not need to know the actual id value for "text/plain".

Details

Each table in the database has several columns handling low-level matters such as security, ownership, and provenance. To hide some of these details in the object model, each `IObject` instance contains an `ome.model.internal.Details` instance.

Details works something like unix's `stat`:

```
/Types/Images>ls -ltrAG
total 0
-rw----- 1 josh 0 2006-01-25 20:40 Image1
-rw----- 1 josh 0 2006-01-25 20:40 Image2
-rw----- 1 josh 0 2006-01-25 20:40 Image3
-rw-r--r-- 1 josh 0 2006-01-25 20:40 Image100
/Types/Images>stat Image1
  File: `Image1'
  Size: 0                Blocks: 0                IO Block: 4096   regular empty file
```

(continues on next page)

(continued from previous page)

```
Device: 1602h/5634d      Inode: 376221      Links: 1
Access: (0600/-rw-----)  Uid: ( 1003/      josh)   Gid: ( 1001/ ome)
Access: 2006-01-25 20:40:30.0000000000 +0100
Modify: 2006-01-25 20:40:30.0000000000 +0100
Change: 2006-01-25 20:40:30.0000000000 +0100
```

though it can also store arbitrary other attributes (meta-metadata, so to speak) about our model instances. See [Dynamic methods](#) below for more information.

The main methods on Details are:

```
Permissions Details.getPermissions();
List Details.getUpdates();
Event Details.getCreationEvent();
Event Details.getUpdateEvent();
Experimenter Details.getOwner();
ExperimenterGroup Details.getGroup();
ExternalInfo getExternalInfo();
```

though some of the methods will return null, if that column is not available for the given object. See [Interfaces](#) below for more information.

Consumers of the API are encouraged to pass around Details instances rather than specifying particulars, like:

```
if (securitySystem.allowLoad(Project.class, project.getDetails())) {}
// and not
if (project.getDetails().getPermissions().isGranted(USER,READ) && project.getDetails().
    ↪getOwner().getId( myId )) {...}
```

This should hopefully save a good deal of re-coding if we move to true ACL rather than the current filesystem-like access control.

Because it is a field on every type, Details is also on the list of keywords in the type language (above).

Interfaces

To help work with the generated objects, several interfaces are added to their “implements” clause:

Property	Applies_to	Interface	Notes
Base			
owner	! global		need sudo
group	! global		need sudo
version	! immutable		
creationEvent	! global		
updateEvent	! global && ! immutable		
permissions			
externalInfo			
Other			
name		Named	
description		Described	
linkedAnnotationList		IAnnotated	

For example, `ome.model.meta.Experimenter` is a “global” type, therefore it has no `Details.owner` field. In order

to create this type of object, you will either need to have admin privileges, or in some cases, use the `ome.api.IAdmin` interface directly (in the case of enums, you will need to use the `ome.api.ITypes` interface).

Inheritance

Inheritance is supported in the object model. The superclass relationships can be defined simply in the mapping files. One example in <https://github.com/ome/omero-model> is the annotation hierarchy in <https://github.com/ome/omero-model/blob/v5.6.13/src/main/resources/mappings/annotations.ome.xml>. Hibernate supports this polymorphism, and will search all subclasses when a superclass is returned. *However*, due to Hibernate's use of bytecode-generated proxies, testing for class equality is not always straightforward.

Hibernate uses CGLIB and Javassist and similar bytecode generation to perform much of its magic. For these bytecode generated objects, the `getClass()` method returns something of the form `ome.model.core.Image\_$_javassist` which cannot be passed back into Hibernate. Instead, we must first parse that class String with `Utils#trueClass()`.

Model report objects

To support the Collection Counts requirement in which users would like to know how many objects are in a collection by owner, it was necessary to add read-only `Map<String, Long>` fields to all objects with links. See the [Collection counts](#) page for more information.

Dynamic methods

Finally, because not all programming fits into the static programming frame, the object model provides several methods for working dynamically with all `IObject` subclasses.

fieldSet / putAt / retrieve

Each model class contains a public final static String for each field in that class (superclass fields are omitted). A copy of all these fields is available through `fieldSet()`. This field identifier can be used in combination with the `putAt` and `retrieve` methods to store arbitrary data in a class instance. Calls to `putAt` / `retrieve` with a string found in `fieldSet` delegate to the traditional getters/setters. Otherwise, the value is stored in lazily-initialized Map (if no data is stored, the map is null).

acceptFilter

An automation of calls to `putAt` / `retrieve` can be achieved by implementing an `ome.util.Filter`. A Filter is a VisitorPattern-like interface which not only visits every field of an object, but also has the chance to replace the field value with an arbitrary other value. Much of the internal functionality in OMERO is achieved through filters.

Limitations

- The filter methods override all standard checks such as `IObject#isLoading` and so null-pointer exceptions etc. may be thrown.
- The types stored in the dynamic map currently do not propagate to the *OMERO.blitz* model objects, since not all `java.lang.Objects` can be converted.

Entity lifecycle

These additions make certain operations on the model objects easier and cleaner, but they do not save the developer from understanding how each object interacts with Hibernate. Each object has a defined lifecycle and it is important to know both the origin (client, server, or backend) as well as its current state to understand what will and can happen with it.

States

Each instance can be found in one of several states. Quickly, they are:

transient

The entity has been created ("`new Image()`") and not yet shown to the backend.

persistent

The entity has been stored in the database and has a non-null id (`IObject.getId()`). Here Hibernate differentiates between detached, managed, and deleted entities. Detached entities do not take part in lazy-loading or dirty detection like managed entities do. They can, however, be re-attached (made “managed”). Deleted entities cannot take part in most of the ORM activities, and exceptions will be thrown if they are encountered.

unloaded (a reference, or proxy)

To solve the common problem of lazy loading exceptions found in many Hibernate applications, we have introduced the concept of unloaded proxy objects which are objects with all fields nulled other than the id. Attempts to get or set any other property will result in an exception. The backend detects these proxies and restores their value before operating on the graph. There are two related states for collections – `null` which is completely unloaded, and `filtered` in which certain items have been removed (more on this below).

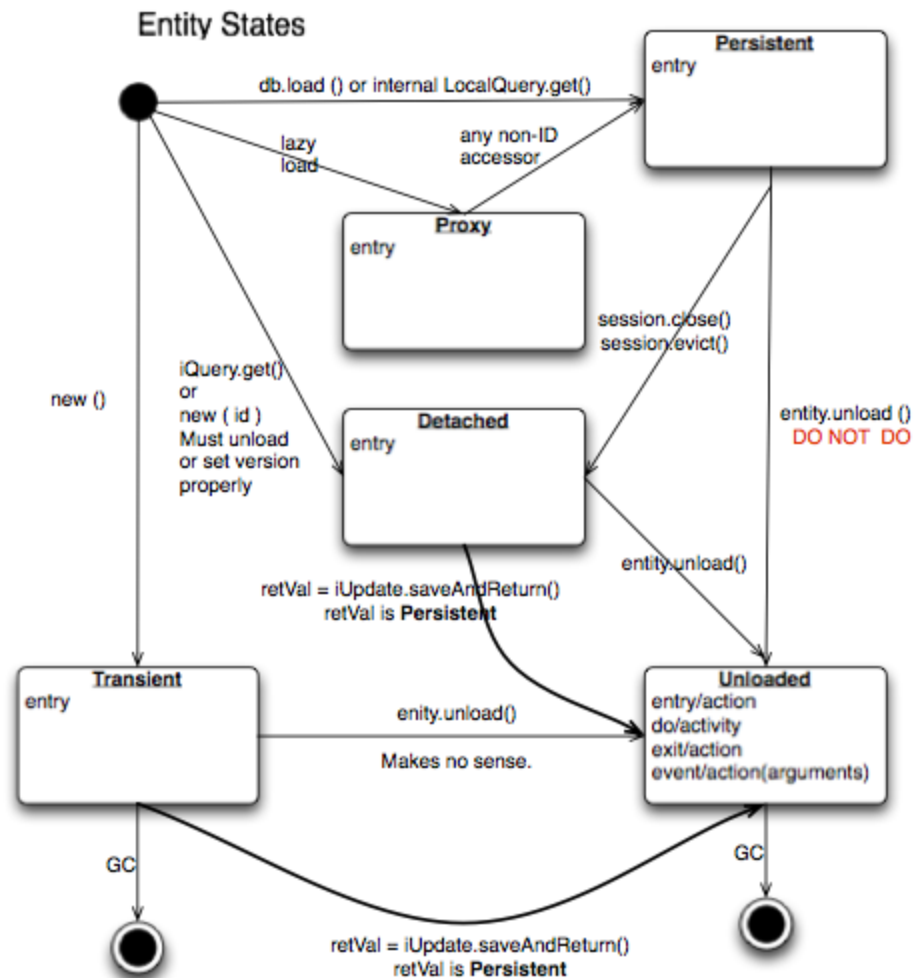
Identity, references, and versions

Critical for understanding these states is understanding the concepts of identity and versioning as it relates to ORM. Every object has an id field that if created by the backend will not be `null`. However, every table does not have a primary key field – subclasses contain a foreign key link to their superclass. Therefore all objects without an id are assumed to be non-persistent (i.e. transient).

Though the id cannot be the sole decider of equality since there are issues with the Java definition of `equals()` and `hashCode()`, we often perform lookups based on the class and id of an instance. Here again caution must be taken not to unintentionally use a possibly bytecode-generated subclass. See the discussion under *Inheritance* above.

Class/id-based lookup is in fact so useful that it is possible to take an model object and call `obj.unload()` to have a “reference” – essentially a placeholder for a model object that contains only an id. Calls to any accessors other than `get/setId` will throw an exception. An object can be tested for loadedness with `obj.isLoading()`.

A client can use unloaded instances to inform the backend that a certain information is not available and should be filled in server-side. For example, a user can do the following:



```
Project p = new Project();
Dataset d = new Dataset( new Long(1), false); // this means create an already unloaded_
↪instance
p.linkDataset(d);
iUpdate.saveObject(p);
```

The server, in turn, also uses references to replace backend proxies that would otherwise throw `LazyInitializationExceptions` on serialization. Clients, therefore, must code with the expectation that the leaves in an object graph may be unloaded. Extending a query with “outer join fetch” will cause these objects to be loaded as well. For example:

```
select p from Project p
    left outer join fetch p.datasetLinks as links
    left outer join fetch links.child as dataset
```

but eventually in the complex OME metadata graph, it is certain that something will remain unloaded.

Versions are the last piece to understanding object identity. Two entities with the same id should not be considered equal if they have differing versions. On each write operation, the version of an entity is incremented. This allows us to perform optimistic locking so that two users do not simultaneously edit the same object. That works so:

1. User A and User B retrieve Object X id=1, version=0.
2. User A edits Object X and saves it. Version is incremented to 1.
3. User B edits Object X and tries to save it. The SQL generated is: UPDATE table SET value = newvalue WHERE id = 1 and version = 0; which updates no rows.
4. The fact that no rows were altered is seen by the backend and an `OptimisticLockException` is thrown.

Identity and versioning make working with the object model difficult sometimes, but guarantee that our data is never corrupted.

Working with the object model

With these states in mind, it is possible to start looking at how to actually use model objects. From the point of view of the server, everything is either an assertion of an object graph (a “write”) or a request for an object graph (a “read”), whether they are coming from an RMI client, an *OMERO.blitz* client, or even being generated internally.

Writing

Creating new objects is as simple as instantiating objects and linking them together. If all NOT-NULL fields are not filled, then a `ValidationException` will be thrown by the server:

```
IUpdate update = new ServiceFactory().getUpdateService();
Image i = new Image();
try {
    update.saveObject(i);
} catch (ValidationException ve) {
    // not ok.
}
i.setName("image");
return update.saveAndReturnObject(i); // ok.
```

Otherwise, the returned value will be the Image with its id field filled. This works on arbitrarily complex graphs of objects:

```
Image i = new Image("image-name"); // This constructor exists because "name" is the only
↳required field.
Dataset d = new Dataset("dataset-name");
TagAnnotation tag = new TagAnnotation();
tag.setTextValue("some-tag");
i.linkDataset(d);
i.linkAnnotation(tag);
update.saveAndReturnObject(i);
```

Reading

Reading is a similarly straightforward operation. From a simple id-based lookup, `iQuery.get(Experimenter.class, 1L)` to a search for an arbitrarily complex graph:

```
Image i = iQuery.findByQuery("select i from Image i "+
    "join fetch i.datasetLinks as dlinks "+
    "join fetch i.annotationLinks as alinks "+
    "join fetch i.details.owner as owner "+
    "join fetch owner.details.creationEvent "+
    "where i.id = :id", new Parameters().addId(1L));
```

In the return graph, you are guaranteed that any two instances of the same class with the same id are the same object. For example:

```
Image i = ...; // From query
Dataset d = i.linkedDatasetList().get(0);
Image i2 = d.linkedImageList().get(0);
if (i.getId().equals(i2.getId())) {
    assert i == i2 : "Instances must be referentially equal";
}
```

Reading and writing

Complications arise when you try to mix objects from different read operations because of the difference in equality. In all but the most straightforward applications, references to `IObject` instances from different return graphs will start to intermingle. For example, when a user logs in, you might query for all Projects belonging to the user:

```
List<Project> projects = iQuery.findAllByQuery("select p from Project p where p.details.
↳owner.omeName = someUser", null);
Project p = projects.get(0);
Long id = p.getId();
```

Later you might query for Datasets, and be returned some of the same Projects again within the same graph. You have now possibly got two versions of the Project with a given id within your application. And if one of those Projects has a new Dataset reference, then Hibernate would not know whether the object should be removed or not.

```
Project oldProject = ...; // Acquired from first query
// Do some other work
```

(continues on next page)

(continued from previous page)

```

Dataset dataset = iQuery.findByQuery("select d from Dataset d "+
    "join fetch d.projectsLinks links "+
    "join fetch links.parent "+
    "where d.id = :id", new Parameters().addId(5L));
Project newProject = dataset.linkedProjectList().get(0);
assert newProject.getId().equals(oldProject.getId()) : "same object";
assert newProject.sizeOfDatasetLinks() == oldProject.sizeOfDatasetLinks() :
    "if this is false, then saving oldProject is a problem";

```

Without optimistic locks, trying to save `oldProject` would cause whatever Datasets were missing from it to be removed from `newProject` as well. Instead, an `OptimisticLockException` is thrown if a user tries to change an older reference to an entity. Similar problems also arise in multi-user settings, when two users try to access the same object, but it is not purely due to multiple users or even multiple threads, but simply due to stale state.

Note: There is an issue with multiple users in which a `SecurityViolation` is thrown instead of an `OptimisticLockException`.

Various techniques to help to manage these duplications are:

- Copy all data to your own model.
- Return unloaded objects wherever possible.
- Be very careful about the operations you commit and about the order they take place in.
- Use a `ClientSession`.

Lazy loading

An issue related to identity is lazy loading. When an object graph is requested, Hibernate loads only the objects which are directly requested. All others are replaced with proxy objects. Within the Hibernate session, these objects are “active” and if accessed, they will be automatically loaded. This is taken care of by the first-level cache, and is also the reason that referential equality is guaranteed within the Hibernate session. Outside of the session however, the proxies can no longer be loaded and so they cannot be serialized to the client.

Instead, as the return value passes through OMERO’s AOP layer, they get disconnected. Single-valued fields are replaced by an unloaded version:

```

OriginalFile ofile = ...; // Object to test
if ( ! Hibernate.initialized( ofile.getFormat() ) {
    ofile.setFormat( new Format( ofile.getFormat().getId(), false) );
}

```

Multi-valued fields, or collections, are simply nulled. In this case, the `sizeOfXXX` method will return a value less than zero:

```

Dataset d = ...; // Dataset obtained from a query. Didn't request Projects
assert d.sizeOfProjects() < 0 : "Projects should not be loaded";

```

This is why it is necessary to specify all “join fetch” clauses for instances which are required on the client-side. See [ProxyCleanupFilter](#) for the implementation.

Collections

More than just the nulling during serialization, collections pose several interesting problems.

For example, a collection may be filtered on retrieval:

```
Dataset d = iQuery.findByQuery("select d from Dataset d "+
    "join fetch d.projectLinks links "+
    "where links.parent.id > 2000", null);
```

Some ProjectDatasetLink instances have been filtered from the projectLinks collection. If the client decides to save this collection back, there is no way to know that it is incomplete, and Hibernate will remove the missing Projects from the Dataset. It is the developer's responsibility to know what state a collection is in. In the case of links, discussed below, one solution is to use the link objects directly, even if they are largely hidden with the API, but the problem remains for 1-N collections.

Links

A special form of links collection model the many-to-many relationship between two other objects. A Project can contain any number of Datasets, and a Dataset can be in any number of Projects. This is achieved by ProjectDatasetLinks, which have a Project "parent" and a Dataset "child" (the parent/child terms are somewhat arbitrary but are intended to fit roughly with the users' expectations for those types).

It is possible to both add and remove a link directly:

```
ProjectDatasetLink link = new ProjectDatasetLink();
link.setParent( someProject );
link.setChild( someDataset );
link = update.saveAndReturnObject( link );

// someDataset is now included in someProject

update.deleteObject(link);
// or update.deleteObject(new ProjectDatasetLink(link.getId(), false)); // a proxy

// Now the Dataset is not included,
// __unless__ there was already another link.
```

However, it is also possible to have the links managed for you:

```
someProject.linkDataset( someDataset ); // This creates the link
update.saveObject( someProject ); // Notices added link, and saves it

someProject.unlinkDataset( someDataset );
update.saveObject( someProject ); // Notices removal, and deletes it
```

The difficulty with this approach is that unlinkDataset() will fail if the someDataset which you are trying to remove is not referentially equal. That is:

```
someProject.linkDataset( someDataset );
updatedProject = update.saveAndReturnObject( someProject );

updatedProject.unlinkDataset( someDataset );
update.saveObject( updateProject ); // will do __nothing__ !
```

does not work since someDataset is not included in updatedProject, but rather updatedDataset with the same id is. Therefore, it would be necessary to do something along the following lines of:

```
updatedProject = ...; // As before
for (Dataset updatedDataset : updatedProject.linkedDatasetList() ) {
    if (updatedDataset.getId().equals( someDataset.getId() )) {
        updatedProject.unlinkDataset( updatedDataset );
    }
}
```

The unlink method in this case, removes the link from both the Project.datasetLinks collection as well as from the Dataset.projectLinks collection. Hibernate notices that both collections are in agreement, and deletes the Project-DatasetLink (this is achieved via the “delete-orphan” annotation in Hibernate). If only one side of the collection has had its link removed, an exception will be thrown.

Synchronization

Another important point is that the model objects are in no way synchronized. All synchronization must occur within application code.

Limitations

We try to minimize differences between the Model as described by the XML specification and its implementation in the OMERO database but some Objects may behave in a more restricted fashion within OMERO. Examples include:

- ROIs and rendering settings can only belong to one Image

3.8.2 Working with annotations

Structured annotations permit the attachment of data and metadata outside the OMERO data model to certain types within the model. The annotations are designed for individualized use by both sites and tools. Annotations can be attached to multiple instances simultaneously to quickly annotate all entities in a view. Each annotation may have a “namespace” (ns) set. Tools can recognize specific namespaces and interpret those annotations accordingly.

Annotated and annotating types

Each type which can be annotated implements `ome.model.IAnnotated`. Currently, these are:

- Project
- Dataset
- Image
- Pixels
- OriginalFile
- PlaneInfo
- Roi
- Channel
- Annotation and all annotation subtypes in order to form hierarchies

- ScreenPlateWell: Screen, ScreenAcquisition, Plate, Well, Reagent
- Folder

Annotation hierarchy

Though they largely are all String or numeric values, a hierarchy of annotations makes differentiating between just what interpretation should be given to the annotation. This may eventually include validation of the input string and/or file.

```

Annotation (A*) ..... A name field and a description
  ListAnnotation ..... Uses AnnotationAnnotation links to form a list,
↳ of annotations
  BasicAnnotation (A*) ..... Literal or "primitive" values
    BooleanAnnotation ..... A simple true/false flag
    TimestampAnnotation ..... A date/time
    TermAnnotation ..... A term used in an ontology
    NumericAnnotation (A*) ..... Floating point and integer values
      DoubleAnnotation
      LongAnnotation
  MapAnnotation ..... A list of key-value pairs
  TextAnnotation (A*) ..... A single text field
    CommentAnnotation ..... A user comment
    TagAnnotation ..... Interpreted as a Web 2.0 "tag" on an object,
↳ tags on tags form tag bundles
    XmlAnnotation ..... An xml snippet attached to some object
  TypeAnnotation (A*) ..... Links some entity to another (possibly to be
↳ replaced by <any/>)
    FileAnnotation ..... Uses the Format field on OriginalFile to specify
↳ type
A* = abstract

```

See also:

Schema documentation for Structured Annotations

Section of the auto-generated schema documentation describing the structured annotations

Names and namespaces

Since arbitrary blobs or clobs can be attached to an entity, it is necessary for clients to have some way to differentiate what it can parse. In many cases, the name might be a simple reminder for a user to find the file s/he has annotated. Applications, however, will most likely want to define a namespace, like `http://name-of-application-provider.com/name-of-application/file-type/version`. Queries can then be produced which search for the proper namespace or match on a part of the name space:

```
iQuery.findAllByQuery("select annotation from FileAnnotation where "+
  "name like 'http://name-of-application-provider.com/name-of-application/%'");
```

Tags will most likely begin without a namespace. As a tag gets escalated to a common vocabulary, it might make sense to add a possibly site-specific namespace with more well-defined semantics.

Descriptions

Unlike the previous, `ImageAnnotation` and `DatasetAnnotation` types, the new structured annotations do not have a description field. The single description field was limited for multi-user scenarios, and can be fully replaced by `TextAnnotations` attached to another annotation.

```
FileAnnotation fileAnnotation = ...;
TextAnnotation description = ...;
fileAnnotation.linkAnnotation(description);
```

Examples

Examples of creating various type of Annotations can also be found on the [Java](#) and [Python](#) pages.

Basics

```
import ome.model.IAnnotated;
import ome.model.annotations.FileAnnotation;
import ome.model.annotations.TagAnnotation;
import ome.model.core.OriginalFile;
import ome.model.display.Roi;

List<Annotation> list = iAnnotated.linkedAnnotationList();
// do something with list
```

Attaching a tag

```
TagAnnotation tag = new TagAnnotation();
tag.setTextValue("interesting");

Roi roi = ...; // Some region of interest
ILink link = roi.linkAnnotation(tag);

iUpdate.saveObject(link);
```

Attaching a file

```
// or attach something new
OriginalFile myOriginalFile = new OriginalFile();
myOriginalFile.setName("output.pdf");
// upload PDF

FileAnnotation annotation = new FileAnnotation();
annotation.setName("http://example.com/myClient/analysisOutput");
annotation.setFile(myOriginalFile);

ILink link = iAnnotated.linkAnnotation(annotation);
link = iUpdate.saveAndReturnObject(link);
```

All write changes are intended to occur through the IUpdate interface, whereas searching should be significantly easier through ome.api.Search than IQuery.

See also:

Extending OMERO.server

Map annotations

3.8.3 Glossary of all OMERO Model Objects

Overview

In navigating the model objects used by the *OMERO API* and in **omero hql** it is often useful to look up the names of the object properties and the types of their values. This reference document lists every OMERO model object and their more useful properties, with an emphasis on enumerating every direct relationship among the objects.

Reference

AcquisitionMode

Used by: *LogicalChannel.mode*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

AdminPrivilege

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

AffineTransform

Used by: *Shape.transform*

Properties:

a00: double
a01: double
a02: double
a10: double
a11: double
a12: double
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*

version: integer (optional), see [IMutable](#)

Annotation

Subclasses: *BasicAnnotation*, *ListAnnotation*, *MapAnnotation*, *TextAnnotation*, *TypeAnnotation*

Used by: *AnnotationAnnotationLink.child*, *AnnotationAnnotationLink.parent*, *ChannelAnnotationLink.child*, *DatasetAnnotationLink.child*, *DetectorAnnotationLink.child*, *DichroicAnnotationLink.child*, *ExperimenterAnnotationLink.child*, *ExperimenterGroupAnnotationLink.child*, *FilesetAnnotationLink.child*, *FilterAnnotationLink.child*, *FolderAnnotationLink.child*, *ImageAnnotationLink.child*, *InstrumentAnnotationLink.child*, *LightPathAnnotationLink.child*, *LightSourceAnnotationLink.child*, *NamespaceAnnotationLink.child*, *NodeAnnotationLink.child*, *ObjectiveAnnotationLink.child*, *OriginalFileAnnotationLink.child*, *PlaneInfoAnnotationLink.child*, *PlateAcquisitionAnnotationLink.child*, *PlateAnnotationLink.child*, *ProjectAnnotationLink.child*, *ReagentAnnotationLink.child*, *RoiAnnotationLink.child*, *ScreenAnnotationLink.child*, *SessionAnnotationLink.child*, *ShapeAnnotationLink.child*, *WellAnnotationLink.child*

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple)
 description: text (optional)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 name: text (optional)
 ns: text (optional)
 version: integer (optional), see [IMutable](#)

AnnotationAnnotationLink

Used by: *Annotation.annotationLinks*

Properties:

child: *Annotation*, see [ILink](#)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Annotation*, see [ILink](#)
 version: integer (optional), see [IMutable](#)

Arc

Properties:

annotationLinks: *LightSourceAnnotationLink* (multiple) from *LightSource*
details.creationEvent: *Event* from *LightSource*
details.externalInfo: *ExternalInfo* (optional) from *LightSource*
details.group: *ExperimenterGroup* from *LightSource*
details.owner: *Experimenter* from *LightSource*
details.updateEvent: *Event* from *LightSource*
instrument: *Instrument* from *LightSource*
lotNumber: **string** (optional) from *LightSource*
manufacturer: **string** (optional) from *LightSource*
model: **string** (optional) from *LightSource*
power.unit: enumeration of *Power* (optional) from *LightSource*
power.value: **double** (optional) from *LightSource*
serialNumber: **string** (optional) from *LightSource*
type: *ArcType*
version: **integer** (optional) from *LightSource*

ArcType

Used by: *Arc.type*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: **string**, see *IEnum*

BasicAnnotation

Subclasses: *BooleanAnnotation*, *NumericAnnotation*, *TermAnnotation*, *TimestampAnnotation*

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: **text** (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
name: **text** (optional) from *Annotation*
ns: **text** (optional) from *Annotation*
version: **integer** (optional) from *Annotation*

Binning

Used by: *DetectorSettings.binning*

Properties:

details.externalInfo: *ExternalInfo* (optional)
 value: string, see *IEnum*

BooleanAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
 boolValue: boolean (optional)
 description: text (optional) from *Annotation*
 details.creationEvent: *Event* from *Annotation*
 details.externalInfo: *ExternalInfo* (optional) from *Annotation*
 details.group: *ExperimenterGroup* from *Annotation*
 details.owner: *Experimenter* from *Annotation*
 details.updateEvent: *Event* from *Annotation*
 name: text (optional) from *Annotation*
 ns: text (optional) from *Annotation*
 version: integer (optional) from *Annotation*

Channel

Used by: *ChannelAnnotationLink.parent*, *LogicalChannel.channels*, *Pixels.channels*

Properties:

alpha: integer (optional)
 annotationLinks: *ChannelAnnotationLink* (multiple)
 blue: integer (optional)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 green: integer (optional)
 logicalChannel: *LogicalChannel*
 lookupTable: text (optional)
 pixels: *Pixels*
 red: integer (optional)
 statsInfo: *StatsInfo* (optional)
 version: integer (optional), see *Immutable*

ChannelAnnotationLink

Used by: *Channel.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Channel*, see *ILink*
version: *integer* (optional), see *IMutable*

ChannelBinding

Used by: *CodomainMapContext.channelBinding*, *RenderingDef.waveRendering*

Properties:

active: *boolean*
alpha: *integer*
blue: *integer*
coefficient: *double*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
family: *Family*
green: *integer*
inputEnd: *double*
inputStart: *double*
lookupTable: *string* (optional)
noiseReduction: *boolean*
red: *integer*
renderingDef: *RenderingDef*
spatialDomainEnhancement: *CodomainMapContext* (multiple)
version: *integer* (optional), see *IMutable*

ChecksumAlgorithm

Used by: *OriginalFile.hasher*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

CodomainMapContext

Subclasses: *ContrastStretchingContext*, *PlaneSlicingContext*, *ReverseIntensityContext*

Used by: *ChannelBinding.spatialDomainEnhancement*

Properties:

channelBinding: *ChannelBinding*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
version: integer (optional), see *IMutable*

CommentAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: text (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
name: text (optional) from *Annotation*
ns: text (optional) from *Annotation*
textValue: text (optional) from *TextAnnotation*
version: integer (optional) from *Annotation*

ContrastMethod

Used by: *LogicalChannel.contrastMethod*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

ContrastStretchingContext

Properties:

channelBinding: *ChannelBinding* from *CodomainMapContext*
details.creationEvent: *Event* from *CodomainMapContext*
details.externalInfo: *ExternalInfo* (optional) from *CodomainMapContext*
details.group: *ExperimenterGroup* from *CodomainMapContext*
details.owner: *Experimenter* from *CodomainMapContext*
details.updateEvent: *Event* from *CodomainMapContext*
version: integer (optional) from *CodomainMapContext*
xend: integer
xstart: integer
yend: integer
ystart: integer

Correction

Used by: *Objective.correction*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

DBPatch

Properties:

currentPatch: integer
currentVersion: string
details.externalInfo: *ExternalInfo* (optional)
finished: timestamp (optional)
message: string (optional)
previousPatch: integer
previousVersion: string

Dataset

Used by: *DatasetAnnotationLink.parent*, *DatasetImageLink.parent*, *ProjectDatasetLink.child*

Properties:

annotationLinks: *DatasetAnnotationLink* (multiple)
description: text (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*

imageLinks: *DatasetImageLink* (multiple)
 name: `text`
 projectLinks: *ProjectDatasetLink* (multiple)
 version: `integer` (optional), see *Immutable*

DatasetAnnotationLink

Used by: *Dataset.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Dataset*, see *ILink*
 version: `integer` (optional), see *Immutable*

DatasetImageLink

Used by: *Dataset.imageLinks*, *Image.datasetLinks*

Properties:

child: *Image*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Dataset*, see *ILink*
 version: `integer` (optional), see *Immutable*

Detector

Used by: *DetectorAnnotationLink.parent*, *DetectorSettings.detector*, *Instrument.detector*

Properties:

amplificationGain: `double` (optional)
 annotationLinks: *DetectorAnnotationLink* (multiple)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 gain: `double` (optional)
 instrument: *Instrument*

lotNumber: string (optional)
manufacturer: string (optional)
model: string (optional)
offsetValue: double (optional)
serialNumber: string (optional)
type: *DetectorType*
version: integer (optional), see *Immutable*
voltage.unit: enumeration of *ElectricPotential* (optional)
voltage.value: double (optional)
zoom: double (optional)

DetectorAnnotationLink

Used by: *Detector.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Detector*, see *ILink*
version: integer (optional), see *Immutable*

DetectorSettings

Used by: *LogicalChannel.detectorSettings*

Properties:

binning: *Binning* (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
detector: *Detector*
gain: double (optional)
integration: integer (optional)
offsetValue: double (optional)
readOutRate.unit: enumeration of *Frequency* (optional)
readOutRate.value: double (optional)
version: integer (optional), see *Immutable*
voltage.unit: enumeration of *ElectricPotential* (optional)
voltage.value: double (optional)
zoom: double (optional)

DetectorType

Used by: *Detector.type*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: `string`, see `IEnum`

Dichroic

Used by: *DichroicAnnotationLink.parent*, *FilterSet.dichroic*, *Instrument.dichroic*, *LightPath.dichroic*

Properties:

annotationLinks: *DichroicAnnotationLink* (multiple)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
instrument: *Instrument*
lotNumber: `string` (optional)
manufacturer: `string` (optional)
model: `string` (optional)
serialNumber: `string` (optional)
version: `integer` (optional), see `IMutable`

DichroicAnnotationLink

Used by: *Dichroic.annotationLinks*

Properties:

child: *Annotation*, see `ILink`
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Dichroic*, see `ILink`
version: `integer` (optional), see `IMutable`

DimensionOrder

Used by: *Pixels.dimensionOrder*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

DoubleAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: text (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
doubleValue: double (optional)
name: text (optional) from *Annotation*
ns: text (optional) from *Annotation*
version: integer (optional) from *Annotation*

Ellipse

Properties:

annotationLinks: *ShapeAnnotationLink* (multiple) from *Shape*
details.creationEvent: *Event* from *Shape*
details.externalInfo: *ExternalInfo* (optional) from *Shape*
details.group: *ExperimenterGroup* from *Shape*
details.owner: *Experimenter* from *Shape*
details.updateEvent: *Event* from *Shape*
fillColor: integer (optional) from *Shape*
fillRule: string (optional) from *Shape*
fontFamily: string (optional) from *Shape*
fontSize.unit: enumeration of *Length* (optional) from *Shape*
fontSize.value: double (optional) from *Shape*
fontStyle: string (optional) from *Shape*
locked: boolean (optional) from *Shape*
radiusX: double (optional)
radiusY: double (optional)
roi: *Roi* from *Shape*
strokeColor: integer (optional) from *Shape*
strokeDashArray: string (optional) from *Shape*
strokeWidth.unit: enumeration of *Length* (optional) from *Shape*
strokeWidth.value: double (optional) from *Shape*

textValue: text (optional)
 theC: integer (optional) from *Shape*
 theT: integer (optional) from *Shape*
 theZ: integer (optional) from *Shape*
 transform: *AffineTransform* (optional) from *Shape*
 version: integer (optional) from *Shape*
 x: double (optional)
 y: double (optional)

Event

Used by: *AffineTransform.details.creationEvent*, *AffineTransform.details.updateEvent*, *Annotation.details.creationEvent*, *Annotation.details.updateEvent*, *AnnotationAnnotationLink.details.creationEvent*, *AnnotationAnnotationLink.details.updateEvent*, *Channel.details.creationEvent*, *Channel.details.updateEvent*, *ChannelAnnotationLink.details.creationEvent*, *ChannelAnnotationLink.details.updateEvent*, *ChannelBinding.details.creationEvent*, *ChannelBinding.details.updateEvent*, *CodomainMapContext.details.creationEvent*, *CodomainMapContext.details.updateEvent*, *Dataset.details.creationEvent*, *Dataset.details.updateEvent*, *DatasetAnnotationLink.details.creationEvent*, *DatasetAnnotationLink.details.updateEvent*, *DatasetImageLink.details.creationEvent*, *DatasetImageLink.details.updateEvent*, *Detector.details.creationEvent*, *Detector.details.updateEvent*, *DetectorAnnotationLink.details.creationEvent*, *DetectorAnnotationLink.details.updateEvent*, *DetectorSettings.details.creationEvent*, *DetectorSettings.details.updateEvent*, *Dichroic.details.creationEvent*, *Dichroic.details.updateEvent*, *DichroicAnnotationLink.details.creationEvent*, *DichroicAnnotationLink.details.updateEvent*, *Event.containingEvent*, *EventLog.event*, *Experiment.details.creationEvent*, *Experiment.details.updateEvent*, *ExperimenterAnnotationLink.details.creationEvent*, *ExperimenterAnnotationLink.details.updateEvent*, *ExperimenterGroupAnnotationLink.details.creationEvent*, *ExperimenterGroupAnnotationLink.details.updateEvent*, *ExternalInfo.details.creationEvent*, *Fileset.details.creationEvent*, *Fileset.details.updateEvent*, *FilesetAnnotationLink.details.creationEvent*, *FilesetAnnotationLink.details.updateEvent*, *FilesetEntry.details.creationEvent*, *FilesetEntry.details.updateEvent*, *FilesetJobLink.details.creationEvent*, *FilesetJobLink.details.updateEvent*, *Filter.details.creationEvent*, *Filter.details.updateEvent*, *FilterAnnotationLink.details.creationEvent*, *FilterAnnotationLink.details.updateEvent*, *FilterSet.details.creationEvent*, *FilterSet.details.updateEvent*, *FilterSetEmissionFilterLink.details.creationEvent*, *FilterSetEmissionFilterLink.details.updateEvent*, *FilterSetExcitationFilterLink.details.creationEvent*, *FilterSetExcitationFilterLink.details.updateEvent*, *Folder.details.creationEvent*, *Folder.details.updateEvent*, *FolderAnnotationLink.details.creationEvent*, *FolderAnnotationLink.details.updateEvent*, *FolderImageLink.details.creationEvent*, *FolderImageLink.details.updateEvent*, *FolderRoiLink.details.creationEvent*, *FolderRoiLink.details.updateEvent*, *Image.details.creationEvent*, *Image.details.updateEvent*, *ImageAnnotationLink.details.creationEvent*, *ImageAnnotationLink.details.updateEvent*, *ImagingEnvironment.details.creationEvent*, *ImagingEnvironment.details.updateEvent*, *Instrument.details.creationEvent*, *Instrument.details.updateEvent*, *InstrumentAnnotationLink.details.creationEvent*, *InstrumentAnnotationLink.details.updateEvent*, *Job.details.creationEvent*, *Job.details.updateEvent*, *JobOriginalFileLink.details.creationEvent*, *JobOriginalFileLink.details.updateEvent*, *LightPath.details.creationEvent*, *LightPath.details.updateEvent*, *LightPathAnnotationLink.details.creationEvent*, *LightPathAnnotationLink.details.updateEvent*, *LightPathEmissionFilterLink.details.creationEvent*, *LightPathEmissionFilterLink.details.updateEvent*, *LightPathExcitationFilterLink.details.creationEvent*, *LightPathExcitationFilterLink.details.updateEvent*, *LightSettings.details.creationEvent*, *LightSettings.details.updateEvent*, *LightSource.details.creationEvent*, *LightSource.details.updateEvent*, *LightSourceAnnotationLink.details.creationEvent*, *LightSourceAnnotationLink.details.updateEvent*, *Link.details.creationEvent*, *Link.details.updateEvent*, *LogicalChannel.details.creationEvent*, *LogicalChannel.details.updateEvent*, *MicrobeamManipulation.details.creationEvent*, *MicrobeamManipulation.details.updateEvent*, *Microscope.details.creationEvent*, *Microscope.details.updateEvent*, *NamespaceAnnotationLink.details.creationEvent*, *NamespaceAnnotationLink.details.updateEvent*, *NodeAnnotationLink.details.creationEvent*, *NodeAnnotationLink.details.updateEvent*, *OTF.details.creationEvent*, *OTF.details.updateEvent*, *Objective.details.creationEvent*, *Objective.details.updateEvent*, *ObjectiveAnnotationLink.details.creationEvent*, *ObjectiveAnnotationLink.details.updateEvent*, *ObjectiveSettings.details.creationEvent*,

ObjectiveSettings.details.updateEvent, *OriginalFile.details.creationEvent*, *OriginalFile.details.updateEvent*,
OriginalFileAnnotationLink.details.creationEvent, *OriginalFileAnnotationLink.details.updateEvent*, *Pix-*
els.details.creationEvent, *Pixels.details.updateEvent*, *PixelsOriginalFileMap.details.creationEvent*, *PixelsOrig-*
inalFileMap.details.updateEvent, *PlaneInfo.details.creationEvent*, *PlaneInfo.details.updateEvent*, *PlaneInfoAn-*
notationLink.details.creationEvent, *PlaneInfoAnnotationLink.details.updateEvent*, *Plate.details.creationEvent*,
Plate.details.updateEvent, *PlateAcquisition.details.creationEvent*, *PlateAcquisition.details.updateEvent*, *PlateAc-*
quisitionAnnotationLink.details.creationEvent, *PlateAcquisitionAnnotationLink.details.updateEvent*, *PlateAn-*
notationLink.details.creationEvent, *PlateAnnotationLink.details.updateEvent*, *Project.details.creationEvent*,
Project.details.updateEvent, *ProjectAnnotationLink.details.creationEvent*, *ProjectAnnotation-*
Link.details.updateEvent, *ProjectDatasetLink.details.creationEvent*, *ProjectDatasetLink.details.updateEvent*,
ProjectionDef.details.creationEvent, *ProjectionDef.details.updateEvent*, *QuantumDef.details.creationEvent*,
QuantumDef.details.updateEvent, *Reagent.details.creationEvent*, *Reagent.details.updateEvent*, *ReagentAnnota-*
tionLink.details.creationEvent, *ReagentAnnotationLink.details.updateEvent*, *RenderingDef.details.creationEvent*,
RenderingDef.details.updateEvent, *Roi.details.creationEvent*, *Roi.details.updateEvent*, *RoiAnnota-*
tionLink.details.creationEvent, *RoiAnnotationLink.details.updateEvent*, *Screen.details.creationEvent*,
Screen.details.updateEvent, *ScreenAnnotationLink.details.creationEvent*, *ScreenAnnotationLink.details.updateEvent*,
ScreenPlateLink.details.creationEvent, *ScreenPlateLink.details.updateEvent*, *Session.events*, *SessionAnno-*
tationLink.details.creationEvent, *SessionAnnotationLink.details.updateEvent*, *Shape.details.creationEvent*,
Shape.details.updateEvent, *ShapeAnnotationLink.details.creationEvent*, *ShapeAnnotationLink.details.updateEvent*,
StageLabel.details.creationEvent, *StageLabel.details.updateEvent*, *StatsInfo.details.creationEvent*,
StatsInfo.details.updateEvent, *Thumbnail.details.creationEvent*, *Thumbnail.details.updateEvent*, *Trans-*
mittanceRange.details.creationEvent, *TransmittanceRange.details.updateEvent*, *Well.details.creationEvent*,
Well.details.updateEvent, *WellAnnotationLink.details.creationEvent*, *WellAnnotationLink.details.updateEvent*,
WellReagentLink.details.creationEvent, *WellReagentLink.details.updateEvent*, *WellSample.details.creationEvent*,
WellSample.details.updateEvent

Properties:

containingEvent: *Event* (optional)
 details.externalInfo: *ExternalInfo* (optional)
 experimenter: *Experimenter*
 experimenterGroup: *ExperimenterGroup*
 logs: *EventLog* (multiple)
 session: *Session*
 status: **string** (optional)
 time: **timestamp**
 type: *EventType*

EventLog

Used by: *Event.logs*

Properties:

action: **string**
 details.externalInfo: *ExternalInfo* (optional)
 entityId: **long**
 entityType: **string**
 event: *Event*

EventType

Used by: *Event.type*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

Experiment

Used by: *Image.experiment*, *MicrobeamManipulation.experiment*

Properties:

description: text (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
microbeamManipulation: *MicrobeamManipulation* (multiple)
type: *ExperimentType*
version: integer (optional), see *IMutable*

ExperimentType

Used by: *Experiment.type*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

Experimenter

Used by: *AffineTransform.details.owner*, *Annotation.details.owner*, *AnnotationAnnotationLink.details.owner*, *Channel.details.owner*, *ChannelAnnotationLink.details.owner*, *ChannelBinding.details.owner*, *CodomainMapContext.details.owner*, *Dataset.details.owner*, *DatasetAnnotationLink.details.owner*, *DatasetImageLink.details.owner*, *Detector.details.owner*, *DetectorAnnotationLink.details.owner*, *DetectorSettings.details.owner*, *Dichroic.details.owner*, *DichroicAnnotationLink.details.owner*, *Event.experimenter*, *Experiment.details.owner*, *ExperimenterAnnotationLink.details.owner*, *ExperimenterAnnotationLink.parent*, *ExperimenterGroupAnnotationLink.details.owner*, *ExternalInfo.details.owner*, *Fileset.details.owner*, *FilesetAnnotationLink.details.owner*, *FilesetEntry.details.owner*, *FilesetJobLink.details.owner*, *Filter.details.owner*, *FilterAnnotationLink.details.owner*, *FilterSet.details.owner*, *FilterSetEmissionFilterLink.details.owner*, *FilterSetExcitationFilterLink.details.owner*, *Folder.details.owner*, *FolderAnnotationLink.details.owner*, *FolderImageLink.details.owner*, *FolderRoiLink.details.owner*, *GroupExperimenterMap.child*, *Image.details.owner*, *ImageAnnotationLink.details.owner*, *ImagingEnvironment.details.owner*, *Instrument.details.owner*, *InstrumentAnnotationLink.details.owner*, *Job.details.owner*, *JobOriginalFileLink.details.owner*, *LightPath.details.owner*, *LightPathAnnotationLink.details.owner*, *LightPathEmissionFilterLink.details.owner*, *LightPathExcitationFilterLink.details.owner*, *LightSettings.details.owner*, *LightSource.details.owner*, *LightSourceAnnotationLink.details.owner*, *Link.details.owner*, *LogicalChannel.details.owner*, *MicrobeamManipulation.details.owner*,

Microscope.details.owner, NamespaceAnnotationLink.details.owner, NodeAnnotationLink.details.owner, OTF.details.owner, Objective.details.owner, ObjectiveAnnotationLink.details.owner, ObjectiveSettings.details.owner, OriginalFile.details.owner, OriginalFileAnnotationLink.details.owner, Pixels.details.owner, PixelsOriginalFileMap.details.owner, PlaneInfo.details.owner, PlaneInfoAnnotationLink.details.owner, Plate.details.owner, PlateAcquisition.details.owner, PlateAcquisitionAnnotationLink.details.owner, PlateAnnotationLink.details.owner, Project.details.owner, ProjectAnnotationLink.details.owner, ProjectDatasetLink.details.owner, ProjectionDef.details.owner, QuantumDef.details.owner, Reagent.details.owner, ReagentAnnotationLink.details.owner, RenderingDef.details.owner, Roi.details.owner, RoiAnnotationLink.details.owner, Screen.details.owner, ScreenAnnotationLink.details.owner, ScreenPlateLink.details.owner, Session.owner, Session.sudoer, SessionAnnotationLink.details.owner, Shape.details.owner, ShapeAnnotationLink.details.owner, ShareMember.child, StageLabel.details.owner, StatsInfo.details.owner, Thumbnail.details.owner, TransmittanceRange.details.owner, Well.details.owner, WellAnnotationLink.details.owner, WellReagentLink.details.owner, WellSample.details.owner

Properties:

annotationLinks: *ExperimenterAnnotationLink* (multiple)
config: list (multiple)
details.externalInfo: *ExternalInfo* (optional)
email: *securestring* (optional)
firstName: *securestring*
groupExperimenterMap: *GroupExperimenterMap* (multiple)
institution: *securestring* (optional)
lastName: *securestring*
ldap: *boolean*
middleName: *securestring* (optional)
omeName: *securestring*
version: *integer* (optional), see *IMutable*

ExperimenterAnnotationLink

Used by: *Experimenter.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Experimenter*, see *ILink*
version: *integer* (optional), see *IMutable*

ExperimenterGroup

Used by: *AffineTransform.details.group*, *Annotation.details.group*, *AnnotationAnnotationLink.details.group*, *Channel.details.group*, *ChannelAnnotationLink.details.group*, *ChannelBinding.details.group*, *CodomainMap-Context.details.group*, *Dataset.details.group*, *DatasetAnnotationLink.details.group*, *DatasetImageLink.details.group*, *Detector.details.group*, *DetectorAnnotationLink.details.group*, *DetectorSettings.details.group*, *Dichroic.details.group*, *DichroicAnnotationLink.details.group*, *Event.experimenterGroup*, *Experiment.details.group*, *ExperimenterAnnotationLink.details.group*, *ExperimenterGroupAnnotationLink.details.group*, *ExperimenterGroupAnnotationLink.parent*, *ExternalInfo.details.group*, *Fileset.details.group*, *FilesetAnnotationLink.details.group*, *FilesetEntry.details.group*, *FilesetJobLink.details.group*, *Filter.details.group*, *FilterAnnotationLink.details.group*, *FilterSet.details.group*, *FilterSetEmissionFilterLink.details.group*, *FilterSetExcitationFilterLink.details.group*, *Folder.details.group*, *FolderAnnotationLink.details.group*, *FolderImageLink.details.group*, *FolderRoiLink.details.group*, *GroupExperimenterMap.parent*, *Image.details.group*, *ImageAnnotationLink.details.group*, *ImagingEnvironment.details.group*, *Instrument.details.group*, *InstrumentAnnotationLink.details.group*, *Job.details.group*, *JobOriginalFileLink.details.group*, *LightPath.details.group*, *LightPathAnnotationLink.details.group*, *LightPathEmissionFilterLink.details.group*, *LightPathExcitationFilterLink.details.group*, *LightSettings.details.group*, *LightSource.details.group*, *LightSourceAnnotationLink.details.group*, *Link.details.group*, *LogicalChannel.details.group*, *MicrobeamManipulation.details.group*, *Microscope.details.group*, *NamespaceAnnotationLink.details.group*, *NodeAnnotationLink.details.group*, *OTF.details.group*, *Objective.details.group*, *ObjectiveAnnotationLink.details.group*, *ObjectiveSettings.details.group*, *OriginalFile.details.group*, *OriginalFileAnnotationLink.details.group*, *Pixels.details.group*, *PixelsOriginalFileMap.details.group*, *PlaneInfo.details.group*, *PlaneInfoAnnotationLink.details.group*, *Plate.details.group*, *PlateAcquisition.details.group*, *PlateAcquisitionAnnotationLink.details.group*, *PlateAnnotationLink.details.group*, *Project.details.group*, *ProjectAnnotationLink.details.group*, *ProjectDatasetLink.details.group*, *ProjectionDef.details.group*, *QuantumDef.details.group*, *Reagent.details.group*, *ReagentAnnotationLink.details.group*, *RenderingDef.details.group*, *Roi.details.group*, *RoiAnnotationLink.details.group*, *Screen.details.group*, *ScreenAnnotationLink.details.group*, *ScreenPlateLink.details.group*, *SessionAnnotationLink.details.group*, *Shape.details.group*, *ShapeAnnotationLink.details.group*, *Share.group*, *StageLabel.details.group*, *StatsInfo.details.group*, *Thumbnail.details.group*, *TransmittanceRange.details.group*, *Well.details.group*, *WellAnnotationLink.details.group*, *WellReagentLink.details.group*, *WellSample.details.group*

Properties:

annotationLinks: *ExperimenterGroupAnnotationLink* (multiple)
 config: list (multiple)
 description: text (optional)
 details.externalInfo: *ExternalInfo* (optional)
 groupExperimenterMap: *GroupExperimenterMap* (multiple)
 ldap: boolean
 name: string
 version: integer (optional), see *IMutable*

ExperimenterGroupAnnotationLink

Used by: *ExperimenterGroup.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*

parent: *ExperimenterGroup*, see *ILink*
 version: *integer* (optional), see *IMutable*

ExternalInfo

Used by: *AcquisitionMode.details.externalInfo*, *AdminPrivilege.details.externalInfo*, *AffineTransform.details.externalInfo*, *Annotation.details.externalInfo*, *AnnotationAnnotationLink.details.externalInfo*, *ArcType.details.externalInfo*, *Binning.details.externalInfo*, *Channel.details.externalInfo*, *ChannelAnnotationLink.details.externalInfo*, *ChannelBinding.details.externalInfo*, *ChecksumAlgorithm.details.externalInfo*, *CodomainMapContext.details.externalInfo*, *ContrastMethod.details.externalInfo*, *Correction.details.externalInfo*, *DBPatch.details.externalInfo*, *Dataset.details.externalInfo*, *DatasetAnnotationLink.details.externalInfo*, *DatasetImageLink.details.externalInfo*, *Detector.details.externalInfo*, *DetectorAnnotationLink.details.externalInfo*, *DetectorSettings.details.externalInfo*, *DetectorType.details.externalInfo*, *Dichroic.details.externalInfo*, *DichroicAnnotationLink.details.externalInfo*, *DimensionOrder.details.externalInfo*, *Event.details.externalInfo*, *EventLog.details.externalInfo*, *EventType.details.externalInfo*, *Experiment.details.externalInfo*, *ExperimentType.details.externalInfo*, *Experimenter.details.externalInfo*, *ExperimenterAnnotationLink.details.externalInfo*, *ExperimenterGroup.details.externalInfo*, *ExperimenterGroupAnnotationLink.details.externalInfo*, *ExternalInfo.details.externalInfo*, *Family.details.externalInfo*, *FilamentType.details.externalInfo*, *Fileset.details.externalInfo*, *FilesetAnnotationLink.details.externalInfo*, *FilesetEntry.details.externalInfo*, *FilesetJobLink.details.externalInfo*, *Filter.details.externalInfo*, *FilterAnnotationLink.details.externalInfo*, *FilterSet.details.externalInfo*, *FilterSetEmissionFilterLink.details.externalInfo*, *FilterSetExcitationFilterLink.details.externalInfo*, *FilterType.details.externalInfo*, *Folder.details.externalInfo*, *FolderAnnotationLink.details.externalInfo*, *FolderImageLink.details.externalInfo*, *FolderRoiLink.details.externalInfo*, *Format.details.externalInfo*, *GroupExperimenterMap.details.externalInfo*, *Illumination.details.externalInfo*, *Image.details.externalInfo*, *ImageAnnotationLink.details.externalInfo*, *ImagingEnvironment.details.externalInfo*, *Immersion.details.externalInfo*, *Instrument.details.externalInfo*, *InstrumentAnnotationLink.details.externalInfo*, *Job.details.externalInfo*, *JobOriginalFileLink.details.externalInfo*, *JobStatus.details.externalInfo*, *LaserMedium.details.externalInfo*, *LaserType.details.externalInfo*, *LightPath.details.externalInfo*, *LightPathAnnotationLink.details.externalInfo*, *LightPathEmissionFilterLink.details.externalInfo*, *LightPathExcitationFilterLink.details.externalInfo*, *LightSettings.details.externalInfo*, *LightSource.details.externalInfo*, *LightSourceAnnotationLink.details.externalInfo*, *Link.details.externalInfo*, *LogicalChannel.details.externalInfo*, *Medium.details.externalInfo*, *MicrobeamManipulation.details.externalInfo*, *MicrobeamManipulationType.details.externalInfo*, *Microscope.details.externalInfo*, *MicroscopeType.details.externalInfo*, *Namespace.details.externalInfo*, *NamespaceAnnotationLink.details.externalInfo*, *Node.details.externalInfo*, *NodeAnnotationLink.details.externalInfo*, *OTF.details.externalInfo*, *Objective.details.externalInfo*, *ObjectiveAnnotationLink.details.externalInfo*, *ObjectiveSettings.details.externalInfo*, *OriginalFile.details.externalInfo*, *OriginalFileAnnotationLink.details.externalInfo*, *PhotometricInterpretation.details.externalInfo*, *Pixels.details.externalInfo*, *PixelsOriginalFileMap.details.externalInfo*, *PixelsType.details.externalInfo*, *PlaneInfo.details.externalInfo*, *PlaneInfoAnnotationLink.details.externalInfo*, *Plate.details.externalInfo*, *PlateAcquisition.details.externalInfo*, *PlateAcquisitionAnnotationLink.details.externalInfo*, *PlateAnnotationLink.details.externalInfo*, *Project.details.externalInfo*, *ProjectAnnotationLink.details.externalInfo*, *ProjectDatasetLink.details.externalInfo*, *ProjectionAxis.details.externalInfo*, *ProjectionDef.details.externalInfo*, *ProjectionType.details.externalInfo*, *Pulse.details.externalInfo*, *QuantumDef.details.externalInfo*, *Reagent.details.externalInfo*, *ReagentAnnotationLink.details.externalInfo*, *RenderingDef.details.externalInfo*, *RenderingModel.details.externalInfo*, *Roi.details.externalInfo*, *RoiAnnotationLink.details.externalInfo*, *Screen.details.externalInfo*, *ScreenAnnotationLink.details.externalInfo*, *ScreenPlateLink.details.externalInfo*, *Session.details.externalInfo*, *SessionAnnotationLink.details.externalInfo*, *Shape.details.externalInfo*, *ShapeAnnotationLink.details.externalInfo*, *ShareMember.details.externalInfo*, *StageLabel.details.externalInfo*, *StatsInfo.details.externalInfo*, *Thumbnail.details.externalInfo*, *TransmittanceRange.details.externalInfo*, *Well.details.externalInfo*, *WellAnnotationLink.details.externalInfo*, *WellReagentLink.details.externalInfo*, *WellSample.details.externalInfo*

Properties:

details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)

details.group: *ExperimenterGroup*
details.owner: *Experimenter*
entityId: long
entityType: string
lsid: string (optional)
uuid: string (optional)

Family

Used by: *ChannelBinding.family*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

Filament

Properties:

annotationLinks: *LightSourceAnnotationLink* (multiple) from *LightSource*
details.creationEvent: *Event* from *LightSource*
details.externalInfo: *ExternalInfo* (optional) from *LightSource*
details.group: *ExperimenterGroup* from *LightSource*
details.owner: *Experimenter* from *LightSource*
details.updateEvent: *Event* from *LightSource*
instrument: *Instrument* from *LightSource*
lotNumber: string (optional) from *LightSource*
manufacturer: string (optional) from *LightSource*
model: string (optional) from *LightSource*
power.unit: enumeration of *Power* (optional) from *LightSource*
power.value: double (optional) from *LightSource*
serialNumber: string (optional) from *LightSource*
type: *FilamentType*
version: integer (optional) from *LightSource*

FilamentType

Used by: *Filament.type*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

FileAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: `text` (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
file: *OriginalFile* (optional)
name: `text` (optional) from *Annotation*
ns: `text` (optional) from *Annotation*
version: `integer` (optional) from *Annotation*

Fileset

Used by: *FilesetAnnotationLink.parent*, *FilesetEntry.fileset*, *FilesetJobLink.parent*, *Image.fileset*

Properties:

annotationLinks: *FilesetAnnotationLink* (multiple)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
images: *Image* (multiple)
jobLinks: *FilesetJobLink* (multiple)
templatePrefix: `text`
usedFiles: *FilesetEntry* (multiple)
version: `integer` (optional), see *Immutable*

FilesetAnnotationLink

Used by: *Fileset.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Fileset*, see *ILink*
version: `integer` (optional), see *Immutable*

FilesetEntry

Used by: *Fileset.usedFiles*, *OriginalFile.filesetEntries*

Properties:

clientPath: `text`
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 fileset: *Fileset*
 originalFile: *OriginalFile*
 version: `integer` (optional), see *Immutable*

FilesetJobLink

Used by: *Fileset.jobLinks*

Properties:

child: *Job*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Fileset*, see *ILink*
 version: `integer` (optional), see *Immutable*

Filter

Used by: *FilterAnnotationLink.parent*, *FilterSetEmissionFilterLink.child*, *FilterSetExcitationFilterLink.child*, *Instrument.filter*, *LightPathEmissionFilterLink.child*, *LightPathExcitationFilterLink.child*

Properties:

annotationLinks: *FilterAnnotationLink* (multiple)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 emissionFilterLink: *FilterSetEmissionFilterLink* (multiple)
 excitationFilterLink: *FilterSetExcitationFilterLink* (multiple)
 filterWheel: `string` (optional)
 instrument: *Instrument*
 lotNumber: `string` (optional)
 manufacturer: `string` (optional)
 model: `string` (optional)

serialNumber: `string` (optional)
transmittanceRange: *TransmittanceRange* (optional)
type: *FilterType* (optional)
version: `integer` (optional), see *Immutable*

FilterAnnotationLink

Used by: *Filter.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Filter*, see *ILink*
version: `integer` (optional), see *Immutable*

FilterSet

Used by: *FilterSetEmissionFilterLink.parent*, *FilterSetExcitationFilterLink.parent*, *Instrument.filterSet*, *LogicalChannel.filterSet*, *OTF.filterSet*

Properties:

details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
dichroic: *Dichroic* (optional)
emissionFilterLink: *FilterSetEmissionFilterLink* (multiple)
excitationFilterLink: *FilterSetExcitationFilterLink* (multiple)
instrument: *Instrument*
lotNumber: `string` (optional)
manufacturer: `string` (optional)
model: `string` (optional)
serialNumber: `string` (optional)
version: `integer` (optional), see *Immutable*

FilterSetEmissionFilterLink

Used by: *Filter.emissionFilterLink*, *FilterSet.emissionFilterLink*

Properties:

child: *Filter*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *FilterSet*, see *ILink*
 version: *integer* (optional), see *Immutable*

FilterSetExcitationFilterLink

Used by: *Filter.excitationFilterLink*, *FilterSet.excitationFilterLink*

Properties:

child: *Filter*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *FilterSet*, see *ILink*
 version: *integer* (optional), see *Immutable*

FilterType

Used by: *Filter.type*

Properties:

details.externalInfo: *ExternalInfo* (optional)
 value: *string*, see *IEnum*

Folder

Used by: *Folder.childFolders*, *Folder.parentFolder*, *FolderAnnotationLink.parent*, *FolderImageLink.parent*, *FolderRoiLink.parent*

Properties:

annotationLinks: *FolderAnnotationLink* (multiple)
 childFolders: *Folder* (multiple)
 description: *text* (optional)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*

details.owner: *Experimenter*
details.updateEvent: *Event*
imageLinks: *FolderImageLink* (multiple)
name: *text*
parentFolder: *Folder* (optional)
roiLinks: *FolderRoiLink* (multiple)
version: *integer* (optional), see *Immutable*

FolderAnnotationLink

Used by: *Folder.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Folder*, see *ILink*
version: *integer* (optional), see *Immutable*

FolderImageLink

Used by: *Folder.imageLinks*, *Image.folderLinks*

Properties:

child: *Image*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Folder*, see *ILink*
version: *integer* (optional), see *Immutable*

FolderRoiLink

Used by: *Folder.roiLinks*, *Roi.folderLinks*

Properties:

child: *Roi*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*

parent: *Folder*, see *ILink*
 version: *integer* (optional), see *Immutable*

Format

Used by: *Image.format*

Properties:

details.externalInfo: *ExternalInfo* (optional)
 value: *string*, see *IEnum*

GenericExcitationSource

Properties:

annotationLinks: *LightSourceAnnotationLink* (multiple) from *LightSource*
 details.creationEvent: *Event* from *LightSource*
 details.externalInfo: *ExternalInfo* (optional) from *LightSource*
 details.group: *ExperimenterGroup* from *LightSource*
 details.owner: *Experimenter* from *LightSource*
 details.updateEvent: *Event* from *LightSource*
 instrument: *Instrument* from *LightSource*
 lotNumber: *string* (optional) from *LightSource*
 manufacturer: *string* (optional) from *LightSource*
 map: *list* (multiple)
 model: *string* (optional) from *LightSource*
 power.unit: enumeration of *Power* (optional) from *LightSource*
 power.value: *double* (optional) from *LightSource*
 serialNumber: *string* (optional) from *LightSource*
 version: *integer* (optional) from *LightSource*

GroupExperimenterMap

Used by: *Experimenter.groupExperimenterMap*, *ExperimenterGroup.groupExperimenterMap*

Properties:

child: *Experimenter*, see *ILink*
 details.externalInfo: *ExternalInfo* (optional)
 owner: *boolean*
 parent: *ExperimenterGroup*, see *ILink*
 version: *integer* (optional), see *Immutable*

Illumination

Used by: *LogicalChannel.illumination*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: `string`, see `IEnum`

Image

Used by: *DatasetImageLink.child*, *Fileset.images*, *FolderImageLink.child*, *ImageAnnotationLink.parent*, *Pixels.image*, *Roi.image*, *WellSample.image*

Properties:

acquisitionDate: `timestamp` (optional)
annotationLinks: *ImageAnnotationLink* (multiple)
archived: `boolean` (optional)
datasetLinks: *DatasetImageLink* (multiple)
description: `text` (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
experiment: *Experiment* (optional)
fileset: *Fileset* (optional)
folderLinks: *FolderImageLink* (multiple)
format: *Format* (optional)
imagingEnvironment: *ImagingEnvironment* (optional)
instrument: *Instrument* (optional)
name: `text`
objectiveSettings: *ObjectiveSettings* (optional)
partial: `boolean` (optional)
pixels: *Pixels* (multiple)
rois: *Roi* (multiple)
series: `integer` (optional)
stageLabel: *StageLabel* (optional)
version: `integer` (optional), see *IMutable*
wellSamples: *WellSample* (multiple)

ImageAnnotationLink

Used by: *Image.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Image*, see *ILink*
version: *integer* (optional), see *IMutable*

ImagingEnvironment

Used by: *Image.imagingEnvironment*

Properties:

airPressure.unit: enumeration of *Pressure* (optional)
airPressure.value: *double* (optional)
co2percent: *double* (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
humidity: *double* (optional)
map: list (multiple)
temperature.unit: enumeration of *Temperature* (optional)
temperature.value: *double* (optional)
version: *integer* (optional), see *IMutable*

Immersion

Used by: *Objective.immersion*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: *string*, see *IEnum*

ImportJob

Properties:

details.creationEvent: *Event* from *Job*
details.externalInfo: *ExternalInfo* (optional) from *Job*
details.group: *ExperimenterGroup* from *Job*
details.owner: *Experimenter* from *Job*
details.updateEvent: *Event* from *Job*
finished: timestamp (optional) from *Job*
groupname: string from *Job*
imageDescription: text
imageName: text
message: string from *Job*
originalFileLinks: *JobOriginalFileLink* (multiple) from *Job*
scheduledFor: timestamp from *Job*
started: timestamp (optional) from *Job*
status: *JobStatus* from *Job*
submitted: timestamp from *Job*
type: string from *Job*
username: string from *Job*
version: integer (optional) from *Job*

Warning: This model object is deprecated.
--

IndexingJob

Properties:

details.creationEvent: *Event* from *Job*
details.externalInfo: *ExternalInfo* (optional) from *Job*
details.group: *ExperimenterGroup* from *Job*
details.owner: *Experimenter* from *Job*
details.updateEvent: *Event* from *Job*
finished: timestamp (optional) from *Job*
groupname: string from *Job*
message: string from *Job*
originalFileLinks: *JobOriginalFileLink* (multiple) from *Job*
scheduledFor: timestamp from *Job*
started: timestamp (optional) from *Job*
status: *JobStatus* from *Job*
submitted: timestamp from *Job*
type: string from *Job*
username: string from *Job*
version: integer (optional) from *Job*

Instrument

Used by: *Detector.instrument*, *Dichroic.instrument*, *Filter.instrument*, *FilterSet.instrument*, *Image.instrument*, *InstrumentAnnotationLink.parent*, *LightSource.instrument*, *OTF.instrument*, *Objective.instrument*

Properties:

annotationLinks: *InstrumentAnnotationLink* (multiple)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 detector: *Detector* (multiple)
 dichroic: *Dichroic* (multiple)
 filter: *Filter* (multiple)
 filterSet: *FilterSet* (multiple)
 lightSource: *LightSource* (multiple)
 microscope: *Microscope* (optional)
 objective: *Objective* (multiple)
 otf: *OTF* (multiple)
 version: integer (optional), see *Immutable*

InstrumentAnnotationLink

Used by: *Instrument.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Instrument*, see *ILink*
 version: integer (optional), see *Immutable*

IntegrityCheckJob

Properties:

details.creationEvent: *Event* from *Job*
 details.externalInfo: *ExternalInfo* (optional) from *Job*
 details.group: *ExperimenterGroup* from *Job*
 details.owner: *Experimenter* from *Job*
 details.updateEvent: *Event* from *Job*
 finished: timestamp (optional) from *Job*
 groupname: string from *Job*
 message: string from *Job*

originalFileLinks: *JobOriginalFileLink* (multiple) from *Job*
scheduledFor: timestamp from *Job*
started: timestamp (optional) from *Job*
status: *JobStatus* from *Job*
submitted: timestamp from *Job*
type: string from *Job*
username: string from *Job*
version: integer (optional) from *Job*

Job

Subclasses: *ImportJob*, *IndexingJob*, *IntegrityCheckJob*, *MetadataImportJob*, *ParseJob*, *PixelDataJob*, *ScriptJob*, *ThumbnailGenerationJob*, *UploadJob*

Used by: *FilesetJobLink.child*, *JobOriginalFileLink.parent*

Properties:

details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
finished: timestamp (optional)
groupname: string
message: string
originalFileLinks: *JobOriginalFileLink* (multiple)
scheduledFor: timestamp
started: timestamp (optional)
status: *JobStatus*
submitted: timestamp
type: string
username: string
version: integer (optional), see *Immutable*

JobOriginalFileLink

Used by: *Job.originalFileLinks*

Properties:

child: *OriginalFile*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Job*, see *ILink*
version: integer (optional), see *Immutable*

JobStatus

Used by: *Job.status*

Properties:

details.externalInfo: *ExternalInfo* (optional)
 value: string, see *IEnum*

Label

Properties:

annotationLinks: *ShapeAnnotationLink* (multiple) from *Shape*
 details.creationEvent: *Event* from *Shape*
 details.externalInfo: *ExternalInfo* (optional) from *Shape*
 details.group: *ExperimenterGroup* from *Shape*
 details.owner: *Experimenter* from *Shape*
 details.updateEvent: *Event* from *Shape*
 fillColor: integer (optional) from *Shape*
 fillRule: string (optional) from *Shape*
 fontFamily: string (optional) from *Shape*
 fontSize.unit: enumeration of *Length* (optional) from *Shape*
 fontSize.value: double (optional) from *Shape*
 fontStyle: string (optional) from *Shape*
 locked: boolean (optional) from *Shape*
 roi: *Roi* from *Shape*
 strokeColor: integer (optional) from *Shape*
 strokeDashArray: string (optional) from *Shape*
 strokeWidth.unit: enumeration of *Length* (optional) from *Shape*
 strokeWidth.value: double (optional) from *Shape*
 textValue: text (optional)
 theC: integer (optional) from *Shape*
 theT: integer (optional) from *Shape*
 theZ: integer (optional) from *Shape*
 transform: *AffineTransform* (optional) from *Shape*
 version: integer (optional) from *Shape*
 x: double (optional)
 y: double (optional)

Laser

Properties:

annotationLinks: *LightSourceAnnotationLink* (multiple) from *LightSource*
details.creationEvent: *Event* from *LightSource*
details.externalInfo: *ExternalInfo* (optional) from *LightSource*
details.group: *ExperimenterGroup* from *LightSource*
details.owner: *Experimenter* from *LightSource*
details.updateEvent: *Event* from *LightSource*
frequencyMultiplication: *integer* (optional)
instrument: *Instrument* from *LightSource*
laserMedium: *LaserMedium*
lotNumber: *string* (optional) from *LightSource*
manufacturer: *string* (optional) from *LightSource*
model: *string* (optional) from *LightSource*
pockelCell: *boolean* (optional)
power.unit: enumeration of *Power* (optional) from *LightSource*
power.value: *double* (optional) from *LightSource*
pulse: *Pulse* (optional)
pump: *LightSource* (optional)
repetitionRate.unit: enumeration of *Frequency* (optional)
repetitionRate.value: *double* (optional)
serialNumber: *string* (optional) from *LightSource*
tuneable: *boolean* (optional)
type: *LaserType*
version: *integer* (optional) from *LightSource*
wavelength.unit: enumeration of *Length* (optional)
wavelength.value: *double* (optional)

LaserMedium

Used by: *Laser.laserMedium*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: *string*, see *IEnum*

LaserType

Used by: *Laser.type*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: *string*, see *IEnum*

LightEmittingDiode

Properties:

annotationLinks: *LightSourceAnnotationLink* (multiple) from *LightSource*
 details.creationEvent: *Event* from *LightSource*
 details.externalInfo: *ExternalInfo* (optional) from *LightSource*
 details.group: *ExperimenterGroup* from *LightSource*
 details.owner: *Experimenter* from *LightSource*
 details.updateEvent: *Event* from *LightSource*
 instrument: *Instrument* from *LightSource*
 lotNumber: *string* (optional) from *LightSource*
 manufacturer: *string* (optional) from *LightSource*
 model: *string* (optional) from *LightSource*
 power.unit: enumeration of *Power* (optional) from *LightSource*
 power.value: *double* (optional) from *LightSource*
 serialNumber: *string* (optional) from *LightSource*
 version: *integer* (optional) from *LightSource*

LightPath

Used by: *LightPathAnnotationLink.parent*, *LightPathEmissionFilterLink.parent*, *LightPathExcitationFilterLink.parent*, *LogicalChannel.lightPath*

Properties:

annotationLinks: *LightPathAnnotationLink* (multiple)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 dichroic: *Dichroic* (optional)
 emissionFilterLink: *LightPathEmissionFilterLink* (multiple)
 excitationFilterLink: *LightPathExcitationFilterLink* (multiple)
 version: *integer* (optional), see *IMutable*

LightPathAnnotationLink

Used by: *LightPath.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *LightPath*, see *ILink*

version: integer (optional), see [IMutable](#)

LightPathEmissionFilterLink

Used by: *LightPath.emissionFilterLink*

Properties:

child: *Filter*, see [ILink](#)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *LightPath*, see [ILink](#)
version: integer (optional), see [IMutable](#)

LightPathExcitationFilterLink

Used by: *LightPath.excitationFilterLink*

Properties:

child: *Filter*, see [ILink](#)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *LightPath*, see [ILink](#)
version: integer (optional), see [IMutable](#)

LightSettings

Used by: *LogicalChannel.lightSourceSettings*, *MicrobeamManipulation.lightSourceSettings*

Properties:

attenuation: double (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
lightSource: *LightSource*
microbeamManipulation: *MicrobeamManipulation* (optional)
version: integer (optional), see [IMutable](#)
wavelength.unit: enumeration of [Length](#) (optional)
wavelength.value: double (optional)

LightSource

Subclasses: *Arc*, *Filament*, *GenericExcitationSource*, *Laser*, *LightEmittingDiode*

Used by: *Instrument.lightSource*, *Laser.pump*, *LightSettings.lightSource*, *LightSourceAnnotationLink.parent*

Properties:

- annotationLinks: *LightSourceAnnotationLink* (multiple)
- details.creationEvent: *Event*
- details.externalInfo: *ExternalInfo* (optional)
- details.group: *ExperimenterGroup*
- details.owner: *Experimenter*
- details.updateEvent: *Event*
- instrument: *Instrument*
- lotNumber: string (optional)
- manufacturer: string (optional)
- model: string (optional)
- power.unit: enumeration of *Power* (optional)
- power.value: double (optional)
- serialNumber: string (optional)
- version: integer (optional), see *Immutable*

LightSourceAnnotationLink

Used by: *LightSource.annotationLinks*

Properties:

- child: *Annotation*, see *ILink*
- details.creationEvent: *Event*
- details.externalInfo: *ExternalInfo* (optional)
- details.group: *ExperimenterGroup*
- details.owner: *Experimenter*
- details.updateEvent: *Event*
- parent: *LightSource*, see *ILink*
- version: integer (optional), see *Immutable*

Line

Properties:

- annotationLinks: *ShapeAnnotationLink* (multiple) from *Shape*
- details.creationEvent: *Event* from *Shape*
- details.externalInfo: *ExternalInfo* (optional) from *Shape*
- details.group: *ExperimenterGroup* from *Shape*
- details.owner: *Experimenter* from *Shape*
- details.updateEvent: *Event* from *Shape*
- fillColor: integer (optional) from *Shape*
- fillRule: string (optional) from *Shape*

fontFamily: string (optional) from *Shape*
fontSize.unit: enumeration of *Length* (optional) from *Shape*
fontSize.value: double (optional) from *Shape*
fontStyle: string (optional) from *Shape*
locked: boolean (optional) from *Shape*
markerEnd: string (optional)
markerStart: string (optional)
roi: *Roi* from *Shape*
strokeColor: integer (optional) from *Shape*
strokeDashArray: string (optional) from *Shape*
strokeWidth.unit: enumeration of *Length* (optional) from *Shape*
strokeWidth.value: double (optional) from *Shape*
textValue: text (optional)
theC: integer (optional) from *Shape*
theT: integer (optional) from *Shape*
theZ: integer (optional) from *Shape*
transform: *AffineTransform* (optional) from *Shape*
version: integer (optional) from *Shape*
x1: double (optional)
x2: double (optional)
y1: double (optional)
y2: double (optional)

Link

Properties:

details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
version: integer (optional), see *IMutable*

ListAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: text (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
name: text (optional) from *Annotation*
ns: text (optional) from *Annotation*
version: integer (optional) from *Annotation*

LogicalChannel

Used by: *Channel.logicalChannel*

Properties:

channels: *Channel* (multiple)
 contrastMethod: *ContrastMethod* (optional)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 detectorSettings: *DetectorSettings* (optional)
 emissionWave.unit: enumeration of *Length* (optional)
 emissionWave.value: *double* (optional)
 excitationWave.unit: enumeration of *Length* (optional)
 excitationWave.value: *double* (optional)
 filterSet: *FilterSet* (optional)
 fluor: *string* (optional)
 illumination: *Illumination* (optional)
 lightPath: *LightPath* (optional)
 lightSourceSettings: *LightSettings* (optional)
 mode: *AcquisitionMode* (optional)
 name: *text* (optional)
 ndFilter: *double* (optional)
 otf: *OTF* (optional)
 photometricInterpretation: *PhotometricInterpretation* (optional)
 pinHoleSize.unit: enumeration of *Length* (optional)
 pinHoleSize.value: *double* (optional)
 pockelCellSetting: *integer* (optional)
 samplesPerPixel: *integer* (optional)
 version: *integer* (optional), see *IMutable*

LongAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
 description: *text* (optional) from *Annotation*
 details.creationEvent: *Event* from *Annotation*
 details.externalInfo: *ExternalInfo* (optional) from *Annotation*
 details.group: *ExperimenterGroup* from *Annotation*
 details.owner: *Experimenter* from *Annotation*
 details.updateEvent: *Event* from *Annotation*
 longValue: *long* (optional)
 name: *text* (optional) from *Annotation*
 ns: *text* (optional) from *Annotation*
 version: *integer* (optional) from *Annotation*

MapAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: `text` (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
mapValue: `list` (multiple)
name: `text` (optional) from *Annotation*
ns: `text` (optional) from *Annotation*
version: `integer` (optional) from *Annotation*

Mask

Properties:

annotationLinks: *ShapeAnnotationLink* (multiple) from *Shape*
bytes: `binary` (optional)
details.creationEvent: *Event* from *Shape*
details.externalInfo: *ExternalInfo* (optional) from *Shape*
details.group: *ExperimenterGroup* from *Shape*
details.owner: *Experimenter* from *Shape*
details.updateEvent: *Event* from *Shape*
fillColor: `integer` (optional) from *Shape*
fillRule: `string` (optional) from *Shape*
fontFamily: `string` (optional) from *Shape*
fontSize.unit: enumeration of *Length* (optional) from *Shape*
fontSize.value: `double` (optional) from *Shape*
fontStyle: `string` (optional) from *Shape*
height: `double` (optional)
locked: `boolean` (optional) from *Shape*
pixels: *Pixels* (optional)
roi: *Roi* from *Shape*
strokeColor: `integer` (optional) from *Shape*
strokeDashArray: `string` (optional) from *Shape*
strokeWidth.unit: enumeration of *Length* (optional) from *Shape*
strokeWidth.value: `double` (optional) from *Shape*
textValue: `text` (optional)
theC: `integer` (optional) from *Shape*
theT: `integer` (optional) from *Shape*
theZ: `integer` (optional) from *Shape*
transform: *AffineTransform* (optional) from *Shape*
version: `integer` (optional) from *Shape*
width: `double` (optional)

x: double (optional)
y: double (optional)

Medium

Used by: *ObjectiveSettings.medium*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

MetadataImportJob

Properties:

details.creationEvent: *Event* from *Job*
details.externalInfo: *ExternalInfo* (optional) from *Job*
details.group: *ExperimenterGroup* from *Job*
details.owner: *Experimenter* from *Job*
details.updateEvent: *Event* from *Job*
finished: timestamp (optional) from *Job*
groupname: string from *Job*
message: string from *Job*
originalFileLinks: *JobOriginalFileLink* (multiple) from *Job*
scheduledFor: timestamp from *Job*
started: timestamp (optional) from *Job*
status: *JobStatus* from *Job*
submitted: timestamp from *Job*
type: string from *Job*
username: string from *Job*
version: integer (optional) from *Job*
versionInfo: list (multiple)

MicrobeamManipulation

Used by: *Experiment.microbeamManipulation*, *LightSettings.microbeamManipulation*

Properties:

description: text (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
experiment: *Experiment*
lightSourceSettings: *LightSettings* (multiple)
type: *MicrobeamManipulationType*

version: integer (optional), see [Immutable](#)

MicrobeamManipulationType

Used by: *MicrobeamManipulation.type*

Properties:

details.externalInfo: [ExternalInfo](#) (optional)
value: string, see [IEnum](#)

Microscope

Used by: *Instrument.microscope*

Properties:

details.creationEvent: [Event](#)
details.externalInfo: [ExternalInfo](#) (optional)
details.group: [ExperimenterGroup](#)
details.owner: [Experimenter](#)
details.updateEvent: [Event](#)
lotNumber: string (optional)
manufacturer: string (optional)
model: string (optional)
serialNumber: string (optional)
type: [MicroscopeType](#)
version: integer (optional), see [Immutable](#)

MicroscopeType

Used by: *Microscope.type*

Properties:

details.externalInfo: [ExternalInfo](#) (optional)
value: string, see [IEnum](#)

Namespace

Used by: *NamespaceAnnotationLink.parent*

Properties:

annotationLinks: [NamespaceAnnotationLink](#) (multiple)
description: text (optional)
details.externalInfo: [ExternalInfo](#) (optional)
display: boolean (optional)
displayName: text (optional)
keywords: list (optional)
multivalued: boolean (optional)

name: `text`
 version: `integer` (optional), see `Immutable`

NamespaceAnnotationLink

Used by: *Namespace.annotationLinks*

Properties:

child: *Annotation*, see `ILink`
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Namespace*, see `ILink`
 version: `integer` (optional), see `Immutable`

Node

Used by: *NodeAnnotationLink.parent*, *Session.node*

Properties:

annotationLinks: *NodeAnnotationLink* (multiple)
 conn: `securestring`
 details.externalInfo: *ExternalInfo* (optional)
 down: `timestamp` (optional)
 scale: `integer` (optional)
 sessions: *Session* (multiple)
 up: `timestamp`
 uuid: `securestring`
 version: `integer` (optional), see `Immutable`

NodeAnnotationLink

Used by: *Node.annotationLinks*

Properties:

child: *Annotation*, see `ILink`
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Node*, see `ILink`
 version: `integer` (optional), see `Immutable`

NumericAnnotation

Subclasses: *DoubleAnnotation*, *LongAnnotation*

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: **text** (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
name: **text** (optional) from *Annotation*
ns: **text** (optional) from *Annotation*
version: **integer** (optional) from *Annotation*

OTF

Used by: *Instrument.otf*, *LogicalChannel.otf*

Properties:

details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
filterSet: *FilterSet* (optional)
instrument: *Instrument*
objective: *Objective*
opticalAxisAveraged: **boolean**
path: **string**
pixelsType: *PixelsType*
sizeX: **integer**
sizeY: **integer**
version: **integer** (optional), see *Immutable*

Objective

Used by: *Instrument.objective*, *OTF.objective*, *ObjectiveAnnotationLink.parent*, *ObjectiveSettings.objective*

Properties:

annotationLinks: *ObjectiveAnnotationLink* (multiple)
calibratedMagnification: **double** (optional)
correction: *Correction*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*

details.owner: *Experimenter*
 details.updateEvent: *Event*
 immersion: *Immersion*
 instrument: *Instrument*
 iris: boolean (optional)
 lensNA: double (optional)
 lotNumber: string (optional)
 manufacturer: string (optional)
 model: string (optional)
 nominalMagnification: double (optional)
 serialNumber: string (optional)
 version: integer (optional), see *Immutable*
 workingDistance.unit: enumeration of *Length* (optional)
 workingDistance.value: double (optional)

ObjectiveAnnotationLink

Used by: *Objective.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Objective*, see *ILink*
 version: integer (optional), see *Immutable*

ObjectiveSettings

Used by: *Image.objectiveSettings*

Properties:

correctionCollar: double (optional)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 medium: *Medium* (optional)
 objective: *Objective*
 refractiveIndex: double (optional)
 version: integer (optional), see *Immutable*

OriginalFile

Used by: *FileAnnotation.file*, *FilesetEntry.originalFile*, *JobOriginalFileLink.child*, *OriginalFileAnnotationLink.parent*, *PixelsOriginalFileMap.parent*, *Roi.source*

Properties:

annotationLinks: *OriginalFileAnnotationLink* (multiple)
atime: `timestamp` (optional)
ctime: `timestamp` (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
filesetEntries: *FilesetEntry* (multiple)
hash: `text` (optional)
hasher: *ChecksumAlgorithm* (optional)
mimetype: `string` (optional)
mtime: `timestamp` (optional)
name: `text`
path: `text`
pixelsFileMaps: *PixelsOriginalFileMap* (multiple)
repo: `string` (optional)
size: `long` (optional)
version: `integer` (optional), see *Immutable*

OriginalFileAnnotationLink

Used by: *OriginalFile.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *OriginalFile*, see *ILink*
version: `integer` (optional), see *Immutable*

ParseJob

Properties:

details.creationEvent: *Event* from *Job*
 details.externalInfo: *ExternalInfo* (optional) from *Job*
 details.group: *ExperimenterGroup* from *Job*
 details.owner: *Experimenter* from *Job*
 details.updateEvent: *Event* from *Job*
 finished: timestamp (optional) from *Job*
 groupname: string from *Job*
 message: string from *Job*
 originalFileLinks: *JobOriginalFileLink* (multiple) from *Job*
 params: binary (optional)
 scheduledFor: timestamp from *Job*
 started: timestamp (optional) from *Job*
 status: *JobStatus* from *Job*
 submitted: timestamp from *Job*
 type: string from *Job*
 username: string from *Job*
 version: integer (optional) from *Job*

Path

Properties:

annotationLinks: *ShapeAnnotationLink* (multiple) from *Shape*
 d: text (optional)
 details.creationEvent: *Event* from *Shape*
 details.externalInfo: *ExternalInfo* (optional) from *Shape*
 details.group: *ExperimenterGroup* from *Shape*
 details.owner: *Experimenter* from *Shape*
 details.updateEvent: *Event* from *Shape*
 fillColor: integer (optional) from *Shape*
 fillRule: string (optional) from *Shape*
 fontFamily: string (optional) from *Shape*
 fontSize.unit: enumeration of *Length* (optional) from *Shape*
 fontSize.value: double (optional) from *Shape*
 fontStyle: string (optional) from *Shape*
 locked: boolean (optional) from *Shape*
 roi: *Roi* from *Shape*
 strokeColor: integer (optional) from *Shape*
 strokeDashArray: string (optional) from *Shape*
 strokeWidth.unit: enumeration of *Length* (optional) from *Shape*
 strokeWidth.value: double (optional) from *Shape*
 textValue: text (optional)
 theC: integer (optional) from *Shape*
 theT: integer (optional) from *Shape*

theZ: integer (optional) from *Shape*
transform: *AffineTransform* (optional) from *Shape*
version: integer (optional) from *Shape*

Warning: This model object is deprecated.

PhotometricInterpretation

Used by: *LogicalChannel.photometricInterpretation*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

PixelDataJob

Properties:

details.creationEvent: *Event* from *Job*
details.externalInfo: *ExternalInfo* (optional) from *Job*
details.group: *ExperimenterGroup* from *Job*
details.owner: *Experimenter* from *Job*
details.updateEvent: *Event* from *Job*
finished: timestamp (optional) from *Job*
groupname: string from *Job*
message: string from *Job*
originalFileLinks: *JobOriginalFileLink* (multiple) from *Job*
scheduledFor: timestamp from *Job*
started: timestamp (optional) from *Job*
status: *JobStatus* from *Job*
submitted: timestamp from *Job*
type: string from *Job*
username: string from *Job*
version: integer (optional) from *Job*

Pixels

Used by: *Channel.pixels*, *Image.pixels*, *Mask.pixels*, *Pixels.relatedTo*, *PixelsOriginalFileMap.child*, *PlaneInfo.pixels*, *RenderingDef.pixels*, *Thumbnail.pixels*

Properties:

channels: *Channel* (multiple)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*

dimensionOrder: *DimensionOrder*
 image: *Image*
 methodology: string (optional)
 physicalSizeX.unit: enumeration of *Length* (optional)
 physicalSizeX.value: double (optional)
 physicalSizeY.unit: enumeration of *Length* (optional)
 physicalSizeY.value: double (optional)
 physicalSizeZ.unit: enumeration of *Length* (optional)
 physicalSizeZ.value: double (optional)
 pixelsFileMaps: *PixelsOriginalFileMap* (multiple)
 pixelsType: *PixelsType*
 planeInfo: *PlaneInfo* (multiple)
 relatedTo: *Pixels* (optional) (deprecated)
 settings: *RenderingDef* (multiple)
 sha1: string
 significantBits: integer (optional)
 sizeC: integer
 sizeT: integer
 sizeX: integer
 sizeY: integer
 sizeZ: integer
 thumbnails: *Thumbnail* (multiple)
 timeIncrement.unit: enumeration of *Time* (optional)
 timeIncrement.value: double (optional)
 version: integer (optional), see *Immutable*
 waveIncrement: integer (optional)
 waveStart: integer (optional)

Warning: This model object has a deprecated property.

PixelsOriginalFileMap

Used by: *OriginalFile.pixelsFileMaps*, *Pixels.pixelsFileMaps*

Properties:

child: *Pixels*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *OriginalFile*, see *ILink*
 version: integer (optional), see *Immutable*

PixelsType

Used by: *OTF.pixelsType*, *Pixels.pixelsType*

Properties:

bitSize: integer (optional)
details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

PlaneInfo

Used by: *Pixels.planeInfo*, *PlaneInfoAnnotationLink.parent*

Properties:

annotationLinks: *PlaneInfoAnnotationLink* (multiple)
deltaT.unit: enumeration of *Time* (optional)
deltaT.value: double (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
exposureTime.unit: enumeration of *Time* (optional)
exposureTime.value: double (optional)
pixels: *Pixels*
positionX.unit: enumeration of *Length* (optional)
positionX.value: double (optional)
positionY.unit: enumeration of *Length* (optional)
positionY.value: double (optional)
positionZ.unit: enumeration of *Length* (optional)
positionZ.value: double (optional)
theC: integer
theT: integer
theZ: integer
version: integer (optional), see *Immutable*

PlaneInfoAnnotationLink

Used by: *PlaneInfo.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*

parent: *PlaneInfo*, see *ILink*
 version: `integer` (optional), see *Immutable*

PlaneSlicingContext

Properties:

channelBinding: *ChannelBinding* from *CodomainMapContext*
 constant: `boolean`
 details.creationEvent: *Event* from *CodomainMapContext*
 details.externalInfo: *ExternalInfo* (optional) from *CodomainMapContext*
 details.group: *ExperimenterGroup* from *CodomainMapContext*
 details.owner: *Experimenter* from *CodomainMapContext*
 details.updateEvent: *Event* from *CodomainMapContext*
 lowerLimit: `integer`
 planePrevious: `integer`
 planeSelected: `integer`
 upperLimit: `integer`
 version: `integer` (optional) from *CodomainMapContext*

Plate

Used by: *PlateAcquisition.plate*, *PlateAnnotationLink.parent*, *ScreenPlateLink.child*, *Well.plate*

Properties:

annotationLinks: *PlateAnnotationLink* (multiple)
 columnNamingConvention: `string` (optional)
 columns: `integer` (optional)
 defaultSample: `integer` (optional)
 description: `text` (optional)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 externalIdentifier: `string` (optional)
 name: `text`
 plateAcquisitions: *PlateAcquisition* (multiple)
 rowNamingConvention: `string` (optional)
 rows: `integer` (optional)
 screenLinks: *ScreenPlateLink* (multiple)
 status: `text` (optional)
 version: `integer` (optional), see *Immutable*
 wellOriginX.unit: enumeration of *Length* (optional)
 wellOriginX.value: `double` (optional)
 wellOriginY.unit: enumeration of *Length* (optional)
 wellOriginY.value: `double` (optional)

wells: *Well* (multiple)

PlateAcquisition

Used by: *Plate.plateAcquisitions*, *PlateAcquisitionAnnotationLink.parent*, *WellSample.plateAcquisition*

Properties:

annotationLinks: *PlateAcquisitionAnnotationLink* (multiple)
description: `text` (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
endTime: `timestamp` (optional)
maximumFieldCount: `integer` (optional)
name: `text` (optional)
plate: *Plate*
startTime: `timestamp` (optional)
version: `integer` (optional), see *Immutable*
wellSample: *WellSample* (multiple)

PlateAcquisitionAnnotationLink

Used by: *PlateAcquisition.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *PlateAcquisition*, see *ILink*
version: `integer` (optional), see *Immutable*

PlateAnnotationLink

Used by: *Plate.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*

parent: *Plate*, see *ILink*
 version: `integer` (optional), see *Immutable*

Point

Properties:

annotationLinks: *ShapeAnnotationLink* (multiple) from *Shape*
 details.creationEvent: *Event* from *Shape*
 details.externalInfo: *ExternalInfo* (optional) from *Shape*
 details.group: *ExperimenterGroup* from *Shape*
 details.owner: *Experimenter* from *Shape*
 details.updateEvent: *Event* from *Shape*
 fillColor: `integer` (optional) from *Shape*
 fillRule: `string` (optional) from *Shape*
 fontFamily: `string` (optional) from *Shape*
 fontSize.unit: enumeration of *Length* (optional) from *Shape*
 fontSize.value: `double` (optional) from *Shape*
 fontStyle: `string` (optional) from *Shape*
 locked: `boolean` (optional) from *Shape*
 roi: *Roi* from *Shape*
 strokeColor: `integer` (optional) from *Shape*
 strokeDashArray: `string` (optional) from *Shape*
 strokeWidth.unit: enumeration of *Length* (optional) from *Shape*
 strokeWidth.value: `double` (optional) from *Shape*
 textValue: `text` (optional)
 theC: `integer` (optional) from *Shape*
 theT: `integer` (optional) from *Shape*
 theZ: `integer` (optional) from *Shape*
 transform: *AffineTransform* (optional) from *Shape*
 version: `integer` (optional) from *Shape*
 x: `double` (optional)
 y: `double` (optional)

Polygon

Properties:

annotationLinks: *ShapeAnnotationLink* (multiple) from *Shape*
 details.creationEvent: *Event* from *Shape*
 details.externalInfo: *ExternalInfo* (optional) from *Shape*
 details.group: *ExperimenterGroup* from *Shape*
 details.owner: *Experimenter* from *Shape*
 details.updateEvent: *Event* from *Shape*
 fillColor: `integer` (optional) from *Shape*
 fillRule: `string` (optional) from *Shape*
 fontFamily: `string` (optional) from *Shape*
 fontSize.unit: enumeration of *Length* (optional) from *Shape*

fontSize.value: double (optional) from *Shape*
fontStyle: string (optional) from *Shape*
locked: boolean (optional) from *Shape*
points: text (optional)
roi: *Roi* from *Shape*
strokeColor: integer (optional) from *Shape*
strokeDashArray: string (optional) from *Shape*
strokeWidth.unit: enumeration of *Length* (optional) from *Shape*
strokeWidth.value: double (optional) from *Shape*
textValue: text (optional)
theC: integer (optional) from *Shape*
theT: integer (optional) from *Shape*
theZ: integer (optional) from *Shape*
transform: *AffineTransform* (optional) from *Shape*
version: integer (optional) from *Shape*

Polyline

Properties:

annotationLinks: *ShapeAnnotationLink* (multiple) from *Shape*
details.creationEvent: *Event* from *Shape*
details.externalInfo: *ExternalInfo* (optional) from *Shape*
details.group: *ExperimenterGroup* from *Shape*
details.owner: *Experimenter* from *Shape*
details.updateEvent: *Event* from *Shape*
fillColor: integer (optional) from *Shape*
fillRule: string (optional) from *Shape*
fontFamily: string (optional) from *Shape*
fontSize.unit: enumeration of *Length* (optional) from *Shape*
fontSize.value: double (optional) from *Shape*
fontStyle: string (optional) from *Shape*
locked: boolean (optional) from *Shape*
markerEnd: string (optional)
markerStart: string (optional)
points: text (optional)
roi: *Roi* from *Shape*
strokeColor: integer (optional) from *Shape*
strokeDashArray: string (optional) from *Shape*
strokeWidth.unit: enumeration of *Length* (optional) from *Shape*
strokeWidth.value: double (optional) from *Shape*
textValue: text (optional)
theC: integer (optional) from *Shape*
theT: integer (optional) from *Shape*
theZ: integer (optional) from *Shape*
transform: *AffineTransform* (optional) from *Shape*
version: integer (optional) from *Shape*

Project

Used by: *ProjectAnnotationLink.parent*, *ProjectDatasetLink.parent*

Properties:

annotationLinks: *ProjectAnnotationLink* (multiple)
datasetLinks: *ProjectDatasetLink* (multiple)
description: `text` (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
name: `text`
version: `integer` (optional), see *Immutable*

ProjectAnnotationLink

Used by: *Project.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Project*, see *ILink*
version: `integer` (optional), see *Immutable*

ProjectDatasetLink

Used by: *Dataset.projectLinks*, *Project.datasetLinks*

Properties:

child: *Dataset*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Project*, see *ILink*
version: `integer` (optional), see *Immutable*

ProjectionAxis

Used by: *ProjectionDef.axis*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

ProjectionDef

Used by: *RenderingDef.projections*

Properties:

active: boolean
axis: *ProjectionAxis*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
endPlane: integer (optional)
renderingDef: *RenderingDef*
startPlane: integer (optional)
stepping: integer (optional)
type: *ProjectionType*
version: integer (optional), see *Immutable*

ProjectionType

Used by: *ProjectionDef.type*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

Pulse

Used by: *Laser.pulse*

Properties:

details.externalInfo: *ExternalInfo* (optional)
value: string, see *IEnum*

QuantumDef

Used by: *RenderingDef.quantization*

Properties:

bitResolution: integer
 cdEnd: integer
 cdStart: integer
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 version: integer (optional), see *Immutable*

Reagent

Used by: *ReagentAnnotationLink.parent*, *Screen.reagents*, *WellReagentLink.child*

Properties:

annotationLinks: *ReagentAnnotationLink* (multiple)
 description: text (optional)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 name: text (optional)
 reagentIdentifier: string (optional)
 screen: *Screen*
 version: integer (optional), see *Immutable*
 wellLinks: *WellReagentLink* (multiple)

ReagentAnnotationLink

Used by: *Reagent.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Reagent*, see *ILink*
 version: integer (optional), see *Immutable*

Rectangle

Properties:

annotationLinks: *ShapeAnnotationLink* (multiple) from *Shape*
details.creationEvent: *Event* from *Shape*
details.externalInfo: *ExternalInfo* (optional) from *Shape*
details.group: *ExperimenterGroup* from *Shape*
details.owner: *Experimenter* from *Shape*
details.updateEvent: *Event* from *Shape*
fillColor: integer (optional) from *Shape*
fillRule: string (optional) from *Shape*
fontFamily: string (optional) from *Shape*
fontSize.unit: enumeration of *Length* (optional) from *Shape*
fontSize.value: double (optional) from *Shape*
fontStyle: string (optional) from *Shape*
height: double (optional)
locked: boolean (optional) from *Shape*
roi: *Roi* from *Shape*
strokeColor: integer (optional) from *Shape*
strokeDashArray: string (optional) from *Shape*
strokeWidth.unit: enumeration of *Length* (optional) from *Shape*
strokeWidth.value: double (optional) from *Shape*
textValue: text (optional)
theC: integer (optional) from *Shape*
theT: integer (optional) from *Shape*
theZ: integer (optional) from *Shape*
transform: *AffineTransform* (optional) from *Shape*
version: integer (optional) from *Shape*
width: double (optional)
x: double (optional)
y: double (optional)

RenderingDef

Used by: *ChannelBinding.renderingDef*, *Pixels.settings*, *ProjectionDef.renderingDef*

Properties:

compression: double (optional)
defaultT: integer
defaultZ: integer
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
model: *RenderingModel*
name: text (optional)

pixels: *Pixels*
 projections: *ProjectionDef* (multiple)
 quantization: *QuantumDef*
 version: `integer` (optional), see *Immutable*
 waveRendering: *ChannelBinding* (multiple)

RenderingModel

Used by: *RenderingDef.model*

Properties:

details.externalInfo: *ExternalInfo* (optional)
 value: `string`, see *IEnum*

ReverseIntensityContext

Properties:

channelBinding: *ChannelBinding* from *CodomainMapContext*
 details.creationEvent: *Event* from *CodomainMapContext*
 details.externalInfo: *ExternalInfo* (optional) from *CodomainMapContext*
 details.group: *ExperimenterGroup* from *CodomainMapContext*
 details.owner: *Experimenter* from *CodomainMapContext*
 details.updateEvent: *Event* from *CodomainMapContext*
 reverse: `boolean`
 version: `integer` (optional) from *CodomainMapContext*

Roi

Used by: *FolderRoiLink.child*, *Image.rois*, *RoiAnnotationLink.parent*, *Shape.roi*

Properties:

annotationLinks: *RoiAnnotationLink* (multiple)
 description: `text` (optional)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 folderLinks: *FolderRoiLink* (multiple)
 image: *Image* (optional)
 name: `text` (optional)
 shapes: *Shape* (multiple)
 source: *OriginalFile* (optional)
 version: `integer` (optional), see *Immutable*

RoiAnnotationLink

Used by: *Roi.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Roi*, see *ILink*
version: *integer* (optional), see *Immutable*

Screen

Used by: *Reagent.screen*, *ScreenAnnotationLink.parent*, *ScreenPlateLink.parent*

Properties:

annotationLinks: *ScreenAnnotationLink* (multiple)
description: *text* (optional)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
name: *text*
plateLinks: *ScreenPlateLink* (multiple)
protocolDescription: *text* (optional)
protocolIdentifier: *string* (optional)
reagentSetDescription: *text* (optional)
reagentSetIdentifier: *string* (optional)
reagents: *Reagent* (multiple)
type: *string* (optional)
version: *integer* (optional), see *Immutable*

ScreenAnnotationLink

Used by: *Screen.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*

parent: *Screen*, see *ILink*
version: integer (optional), see *Immutable*

ScreenPlateLink

Used by: *Plate.screenLinks*, *Screen.plateLinks*

Properties:

child: *Plate*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Screen*, see *ILink*
version: integer (optional), see *Immutable*

ScriptJob

Properties:

description: string (optional)
details.creationEvent: *Event* from *Job*
details.externalInfo: *ExternalInfo* (optional) from *Job*
details.group: *ExperimenterGroup* from *Job*
details.owner: *Experimenter* from *Job*
details.updateEvent: *Event* from *Job*
finished: timestamp (optional) from *Job*
groupname: string from *Job*
message: string from *Job*
originalFileLinks: *JobOriginalFileLink* (multiple) from *Job*
scheduledFor: timestamp from *Job*
started: timestamp (optional) from *Job*
status: *JobStatus* from *Job*
submitted: timestamp from *Job*
type: string from *Job*
username: string from *Job*
version: integer (optional) from *Job*

Session

Subclasses: *Share*

Used by: *Event.session*, *Node.sessions*, *SessionAnnotationLink.parent*

Properties:

annotationLinks: *SessionAnnotationLink* (multiple)
closed: *timestamp* (optional)
defaultEventType: *string*
details.externalInfo: *ExternalInfo* (optional)
events: *Event* (multiple)
message: *text* (optional)
node: *Node*
owner: *Experimenter*
started: *timestamp*
sudoer: *Experimenter* (optional)
timeToIdle: *long*
timeToLive: *long*
userAgent: *string* (optional)
uuid: *securestring*
version: *integer* (optional), see *Immutable*

SessionAnnotationLink

Used by: *Session.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Session*, see *ILink*
version: *integer* (optional), see *Immutable*

Shape

Subclasses: *Ellipse*, *Label*, *Line*, *Mask*, *Path*, *Point*, *Polygon*, *Polyline*, *Rectangle*

Used by: *Roi.shapes*, *ShapeAnnotationLink.parent*

Properties:

annotationLinks: *ShapeAnnotationLink* (multiple)
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*

details.updateEvent: *Event*
 fillColor: integer (optional)
 fillRule: string (optional)
 fontFamily: string (optional)
 fontSize.unit: enumeration of *Length* (optional)
 fontSize.value: double (optional)
 fontStyle: string (optional)
 locked: boolean (optional)
 roi: *Roi*
 strokeColor: integer (optional)
 strokeDashArray: string (optional)
 strokeWidth.unit: enumeration of *Length* (optional)
 strokeWidth.value: double (optional)
 theC: integer (optional)
 theT: integer (optional)
 theZ: integer (optional)
 transform: *AffineTransform* (optional)
 version: integer (optional), see *Immutable*

ShapeAnnotationLink

Used by: *Shape.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 parent: *Shape*, see *ILink*
 version: integer (optional), see *Immutable*

Share

Used by: *ShareMember.parent*

Properties:

active: boolean
 annotationLinks: *SessionAnnotationLink* (multiple) from *Session*
 closed: timestamp (optional) from *Session*
 data: binary
 defaultEventType: string from *Session*
 details.externalInfo: *ExternalInfo* (optional) from *Session*
 events: *Event* (multiple) from *Session*
 group: *ExperimenterGroup*
 itemCount: long

message: text (optional) from *Session*
node: *Node* from *Session*
owner: *Experimenter* from *Session*
started: timestamp from *Session*
sdoer: *Experimenter* (optional) from *Session*
timeToIdle: long from *Session*
timeToLive: long from *Session*
userAgent: string (optional) from *Session*
uuid: securestring from *Session*
version: integer (optional) from *Session*

ShareMember

Properties:

child: *Experimenter*, see *ILink*
details.externalInfo: *ExternalInfo* (optional)
parent: *Share*, see *ILink*
version: integer (optional), see *IMutable*

StageLabel

Used by: *Image.stageLabel*

Properties:

details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
name: text
positionX.unit: enumeration of *Length* (optional)
positionX.value: double (optional)
positionY.unit: enumeration of *Length* (optional)
positionY.value: double (optional)
positionZ.unit: enumeration of *Length* (optional)
positionZ.value: double (optional)
version: integer (optional), see *IMutable*

StatsInfo

Used by: *Channel.statsInfo*

Properties:

details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 globalMax: double
 globalMin: double
 version: integer (optional), see *Immutable*

TagAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
 description: text (optional) from *Annotation*
 details.creationEvent: *Event* from *Annotation*
 details.externalInfo: *ExternalInfo* (optional) from *Annotation*
 details.group: *ExperimenterGroup* from *Annotation*
 details.owner: *Experimenter* from *Annotation*
 details.updateEvent: *Event* from *Annotation*
 name: text (optional) from *Annotation*
 ns: text (optional) from *Annotation*
 textValue: text (optional) from *TextAnnotation*
 version: integer (optional) from *Annotation*

TermAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
 description: text (optional) from *Annotation*
 details.creationEvent: *Event* from *Annotation*
 details.externalInfo: *ExternalInfo* (optional) from *Annotation*
 details.group: *ExperimenterGroup* from *Annotation*
 details.owner: *Experimenter* from *Annotation*
 details.updateEvent: *Event* from *Annotation*
 name: text (optional) from *Annotation*
 ns: text (optional) from *Annotation*
 termValue: text (optional)
 version: integer (optional) from *Annotation*

TextAnnotation

Subclasses: *CommentAnnotation*, *TagAnnotation*, *XmlAnnotation*

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: text (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
name: text (optional) from *Annotation*
ns: text (optional) from *Annotation*
textValue: text (optional)
version: integer (optional) from *Annotation*

Thumbnail

Used by: *Pixels.thumbnails*

Properties:

details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
mimeType: string
pixels: *Pixels*
ref: string (optional)
sizeX: integer
sizeY: integer
version: integer (optional), see *Immutable*

ThumbnailGenerationJob

Properties:

details.creationEvent: *Event* from *Job*
details.externalInfo: *ExternalInfo* (optional) from *Job*
details.group: *ExperimenterGroup* from *Job*
details.owner: *Experimenter* from *Job*
details.updateEvent: *Event* from *Job*
finished: timestamp (optional) from *Job*
groupname: string from *Job*
message: string from *Job*
originalFileLinks: *JobOriginalFileLink* (multiple) from *Job*
scheduledFor: timestamp from *Job*

started: timestamp (optional) from *Job*
 status: *JobStatus* from *Job*
 submitted: timestamp from *Job*
 type: string from *Job*
 username: string from *Job*
 version: integer (optional) from *Job*

TimestampAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
 description: text (optional) from *Annotation*
 details.creationEvent: *Event* from *Annotation*
 details.externalInfo: *ExternalInfo* (optional) from *Annotation*
 details.group: *ExperimenterGroup* from *Annotation*
 details.owner: *Experimenter* from *Annotation*
 details.updateEvent: *Event* from *Annotation*
 name: text (optional) from *Annotation*
 ns: text (optional) from *Annotation*
 timeValue: timestamp (optional)
 version: integer (optional) from *Annotation*

TransmittanceRange

Used by: *Filter.transmittanceRange*

Properties:

cutIn.unit: enumeration of *Length* (optional)
 cutIn.value: double (optional)
 cutInTolerance.unit: enumeration of *Length* (optional)
 cutInTolerance.value: double (optional)
 cutOut.unit: enumeration of *Length* (optional)
 cutOut.value: double (optional)
 cutOutTolerance.unit: enumeration of *Length* (optional)
 cutOutTolerance.value: double (optional)
 details.creationEvent: *Event*
 details.externalInfo: *ExternalInfo* (optional)
 details.group: *ExperimenterGroup*
 details.owner: *Experimenter*
 details.updateEvent: *Event*
 transmittance: double (optional)
 version: integer (optional), see *Immutable*

TypeAnnotation

Subclasses: *FileAnnotation*

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: text (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
name: text (optional) from *Annotation*
ns: text (optional) from *Annotation*
version: integer (optional) from *Annotation*

UploadJob

Properties:

details.creationEvent: *Event* from *Job*
details.externalInfo: *ExternalInfo* (optional) from *Job*
details.group: *ExperimenterGroup* from *Job*
details.owner: *Experimenter* from *Job*
details.updateEvent: *Event* from *Job*
finished: timestamp (optional) from *Job*
groupname: string from *Job*
message: string from *Job*
originalFileLinks: *JobOriginalFileLink* (multiple) from *Job*
scheduledFor: timestamp from *Job*
started: timestamp (optional) from *Job*
status: *JobStatus* from *Job*
submitted: timestamp from *Job*
type: string from *Job*
username: string from *Job*
version: integer (optional) from *Job*
versionInfo: list (multiple)

Well

Used by: *Plate.wells*, *WellAnnotationLink.parent*, *WellReagentLink.parent*, *WellSample.well*

Properties:

alpha: integer (optional)
annotationLinks: *WellAnnotationLink* (multiple)
blue: integer (optional)
column: integer (optional)
details.creationEvent: *Event*

details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
externalDescription: **text** (optional)
externalIdentifier: **string** (optional)
green: **integer** (optional)
plate: *Plate*
reagentLinks: *WellReagentLink* (multiple)
red: **integer** (optional)
row: **integer** (optional)
status: **string** (optional)
type: **string** (optional)
version: **integer** (optional), see *Immutable*
wellSamples: *WellSample* (multiple)

WellAnnotationLink

Used by: *Well.annotationLinks*

Properties:

child: *Annotation*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Well*, see *ILink*
version: **integer** (optional), see *Immutable*

WellReagentLink

Used by: *Reagent.wellLinks*, *Well.reagentLinks*

Properties:

child: *Reagent*, see *ILink*
details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
parent: *Well*, see *ILink*
version: **integer** (optional), see *Immutable*

WellSample

Used by: *Image.wellSamples*, *PlateAcquisition.wellSample*, *Well.wellSamples*

Properties:

details.creationEvent: *Event*
details.externalInfo: *ExternalInfo* (optional)
details.group: *ExperimenterGroup*
details.owner: *Experimenter*
details.updateEvent: *Event*
image: *Image*
plateAcquisition: *PlateAcquisition* (optional)
posX.unit: enumeration of *Length* (optional)
posX.value: double (optional)
posY.unit: enumeration of *Length* (optional)
posY.value: double (optional)
timepoint: *timestamp* (optional)
version: integer (optional), see *IMutable*
well: *Well*

XmlAnnotation

Properties:

annotationLinks: *AnnotationAnnotationLink* (multiple) from *Annotation*
description: text (optional) from *Annotation*
details.creationEvent: *Event* from *Annotation*
details.externalInfo: *ExternalInfo* (optional) from *Annotation*
details.group: *ExperimenterGroup* from *Annotation*
details.owner: *Experimenter* from *Annotation*
details.updateEvent: *Event* from *Annotation*
name: text (optional) from *Annotation*
ns: text (optional) from *Annotation*
textValue: text (optional) from *TextAnnotation*
version: integer (optional) from *Annotation*

3.8.4 Units

A number of properties in the OME model are physical measurements which inherently have a unit associated with them. Earlier versions of OME defined a default unit for the measurement. Now users, clients, and acquisition systems can specify the unit themselves rather than converting their internal unit to the OME default.

See also:

[Data Model documentation for units](#)

Supported units

- Electric potential
- Frequency
- Length
- Power
- Pressure
- Temperature
- Time

Each of the supported units contain a number of values from the International System of Units (SI) in the most common prefixes from yotta (10^{24}) to yocto (10^{-24}). The Length unit also contains values from the Imperial system as well as internal values which are internal to OME: reference frame and pixel.

Unit fields

The following fields in the OMERO model have a unit type:

Class	Field	Type	Default value
Channel	EmissionWavelength	Length	nm
Channel	ExcitationWavelength	Length	nm
Channel	PinholeSize	Length	μm
Detector	Voltage	ElectricPotential	V
DetectorSettings	ReadOutRate	Frequency	MHz
DetectorSettings	Voltage	ElectricPotential	V
ImagingEnvironment	AirPressure	Pressure	mbar
ImagingEnvironment	Temperature	Temperature	$^{\circ}\text{C}$
Laser	RepetitionRate	Frequency	Hz
Laser	Wavelength	Length	nm
LightSource	Power	Power	mW
LightSourceSettings	Wavelength	Length	nm
Objective	WorkingDistance	Length	μm
Pixels	PhysicalSizeX	Length	μm
Pixels	PhysicalSizeY	Length	μm
Pixels	PhysicalSizeZ	Length	μm
Pixels	TimeIncrement	Time	s
Plane	DeltaT	Time	s
Plane	ExposureTime	Time	s
Plane	PositionX	Length	reference frame
Plane	PositionY	Length	reference frame
Plane	PositionZ	Length	reference frame
Plate	WellOriginX	Length	reference frame
Plate	WellOriginY	Length	reference frame
Shape	FontSize	Length	pt
Shape	StrokeWidth	Length	pixel
StageLabel	X	Length	reference frame
StageLabel	Y	Length	reference frame
StageLabel	Z	Length	reference frame
TransmittanceRange	CutIn	Length	nm

continues on next page

Table 1 – continued from previous page

Class	Field	Type	Default value
TransmittanceRange	CutInTolerance	Length	nm
TransmittanceRange	CutOut	Length	nm
TransmittanceRange	CutOutTolerance	Length	nm
WellSample	PositionX	Length	reference frame
WellSample	PositionY	Length	reference frame

Units of type *reference frame* cannot be assumed comparable to units from other properties including those with type *reference frame*.

Unit objects

Each unit quantity consists of a double-precision scalar and an enumeration which chooses one of the pre-defined values from the model. In code, uppercase spellings of the enumerations are used, while in the schema, in OME-XML files, and in the database, the Unicode symbol for the unit is used.

Language	Representation
Ice	<code>enum UnitsLength { MICROM, ... };</code>
Java and Python	<code>omero.model.enums.UnitsLength.MICROM</code>
C++	<code>omero::model::enums::MICROM</code>
PostgreSQL	<code>'µm'::unitslength</code>

Defining a unit

```
Pixels p = ...; // Defined elsewhere
Length l = new LengthI(2.1, UnitsLength.MICROM); // µm
p.setPhysicalSizeX(l);
p.setPhysicalSizeY(l);
iUpdatePrx.saveObject(p);
```

The above stores a Pixels object in the database with X and Y physical lengths of “µm”.

Converting a unit

Often a measurement will not be in the most convenient unit for display, e.g. 0.00001 mm. could better be expressed in microns. In order to convert between units, pass the measurement that you have available to a constructor of the same type, passing in the target unit that you would like to see:

```
Pixels p = ...; // As saved above
Length l1 = p.getPhysicalSizeX(); // 2.1 microns
Length l2 = new LengthI(x1, UnitsLength.NM); // As nanometers
```

Getting a symbol

The enumerations used in the “units” field of each measurement is of type *omero.model.enums.UnitsNAME* where NAME is *Length*, *Temperature*, etc. These members of that enumeration are all uppercased, code-safe versions of the unit name. To get the symbol as defined in the SI specification, for example, use the *getSymbol* method:

```
Length l1 = ...; // As above
l1.getSymbol(); // Returns "μm"
```

Querying units

In HQL queries, the scalar and the enumeration value can be separately retrieved.

```
select planeInfo.exposureTime.value from PlaneInfo planeInfo ...
```

will retrieve just the double scalar value while

```
select planeInfo.exposureTime.unit from PlaneInfo planeInfo ...
```

will retrieve a string representation of the enum which can be used in each language to create an enum object, e.g.:

```
UnitsTime.valueOf(unit); // Java
getattr(UnitsTime, unit) # Python
```

To load the symbolic representation of the enum which is used internally in the database and is more concise, use an HQL cast:

```
select cast(planeInfo.exposureTime.unit as text) from PlaneInfo planeInfo ...
```

Returning the entire unit quantity will result in a hash map with the various representations:

```
select planeInfo.exposureTime from PlaneInfo planeInfo ...
{'symbol': 's', 'unit': 'SECOND', 'value': 1.2000000476837158}
```

See also:

- https://en.wikipedia.org/wiki/Units_of_measurement
- https://en.wikipedia.org/wiki/System_of_measurement
- https://en.wikipedia.org/wiki/International_System_of_Units

3.8.5 Map annotations

Ongoing advancements in microscopic techniques, analysis systems, and other areas for which OME attempts to provide metadata exchange require a certain flexibility in the OME model. There is a need to store instrument configuration, script parameters, and similar for later use.

Basic text representations are too difficult to parse to be of significant use. An XML format can be more easily parsed, but a single format would have to be agreed upon by all users. Therefore, it is useful to add particular extension points to the model for which no consensus on a data format has been reached, but where more structure than just text is needed.

A *hash map* was the most likely candidate for this data; however, rather than limit this to traditional associative arrays, OME maps are defined as a “**ordered list of key-value pairs**”. The benefit of this representation is that configuration

files which are standard for Java `java.util.Properties` objects can be represented fully in a single map. Duplicates are maintained since there is no unique constraint on the list of pairs.

In most cases, the interpretation of the ordered list will be such that the final value for a particular key “wins” as if each value were placed into a hash map in order, with duplicate values replacing previous ones.

OME-XML

In the OME-XML model, these maps are represented in a compact format. Any map field is defined by the `MapPairs` complex type which consists of `M` elements of the form:

```
<M K="key">value</M>
```

OMERO languages

In OMERO, a slightly more verbose representation of these objects is used. Each map type consists of a list or vector in the respective language, composed of `NamedValue` objects and possibly nulls.

```
// OMERO.java
ImagingEnvironment environment = new ImagingEnvironmentI();
environment.setMap(new List<NamedValue>());
environment.getMap().add(new NamedValue("altitude", "1000m"));
image.setImagingEnvironment(environment);
```

Fields

The concrete fields which are present in the model are currently:

- `ExperimenterGroup.config`
- `GenericExcitationSource.map`
- `ImagingEnvironment.map`
- `ImportJob.versionInfo`

More will be added as demand increases.

MapAnnotations

In addition to the fields above, there is also a *structured annotation* which contains a key-valued pair, the `MapAnnotation`.

```
// OMERO.cpp
MapAnnotation ann = new MapAnnotationI();
ann->getMapValue().push_back(new NamedValueI("run", "5.0"));
ann->getMapValue().push_back(new NamedValueI("run", "4.9"));
ann->getMapValue().push_back(new NamedValueI("run", "5.1"));
```

This permits the flexible attachment of key-value pairs to any of the OME types which are annotatable. Such annotations attached to key UI elements like images and datasets will be presented by the clients, and can be edited with the appropriate permissions. See the section on [Data Annotation](#) in the User guides for more information. See examples of creating `MapAnnotations` in [Java](#) and [Python](#) pages.

Storage and queries

Each map-based field in the OMERO model is represented by an extra table of the form *\$className_\$fieldName*. For example, `MapAnnotation.mapValue` becomes *annotation.mapValue*, where the loss of “map” from the class name is due to the subclassing of *Annotation* by *MapAnnotation*.

In general, use of the specific tables is not necessary and it suffices to write HQL queries based on the classes and field names themselves.

Find the value for a key

```
select nv.value from MapAnnotation ann
  join ann.mapValue as nv
 where nv.name = 'altitude'
```

Finding objects with a key

```
select ann from MapAnnotation ann
  join ann.mapValue as nv
 where nv.name = 'altitude'
```

Finding objects without a key

```
select ann from MapAnnotation ann
 where not exists(
   from MapAnnotation m2
   join m2.mapValue as nv2
   where nv2.name like 'size%')
```

Finding objects with multiple values

```
select ann from MapAnnotation ann
  join ann.mapValue as nv1
  join ann.mapValue as nv2
 where nv1.name = 'date'
       and nv2.name = 'owner'
```

3.8.6 Available transformations

Available transforms	Direction	Status
2003-FC-to-2007-06.xsl	upgrade	excellent
2003-FC-to-2008-09.xsl	upgrade	excellent
2007-06-to-2008-02.xsl	upgrade	excellent
2007-06-to-2008-09.xsl	upgrade	excellent
2008-02-to-2008-09.xsl	upgrade	excellent
2008-09-to-2009-09.xsl	upgrade	excellent
2009-09-to-2010-04.xsl	upgrade	excellent
2010-04-to-2010-06.xsl	upgrade	excellent
2010-06-to-2011-06.xsl	upgrade	excellent
2011-06-to-2012-06.xsl	upgrade	excellent
2012-06-to-2013-06.xsl	upgrade	excellent
2013-06-to-2015-01.xsl	upgrade	excellent
2010-06-to-2003-FC.xsl	downgrade	poor (very lossy)
2010-06-to-2008-02.xsl	downgrade	fair (lossy)
2011-06-to-2010-06.xsl	downgrade	good
2012-06-to-2011-06.xsl	downgrade	good
2013-06-to-2012-06.xsl	downgrade	good
2015-01-to-2013-06.xsl	downgrade	good

Quality of transformations

Source	Targets											
	/2003-FC/	/2007-06/	/2008-02/	/2008-09/	/2009-09/	/2010-04/	/2010-06/	/2011-06/	/2012-06/	/2013-06/	/2015-01/	
/2003-FC/	—	excellent	excellent	excellent	excellent	excellent	excellent	excellent	excellent	excellent	excellent	Upgrades
/2007-06/	poor	—	excellent	excellent	excellent	excellent	excellent	excellent	excellent	excellent	excellent	
/2008-02/	poor	poor	—	excellent	excellent	excellent	excellent	excellent	excellent	excellent	excellent	
/2008-09/	poor	poor	poor	—	excellent	excellent	excellent	excellent	excellent	excellent	excellent	
/2009-09/	poor	poor	poor	poor	—	excellent	excellent	excellent	excellent	excellent	excellent	
/2010-04/	poor	poor	poor	fair	fair	—	excellent	excellent	excellent	excellent	excellent	
/2010-06/	poor	poor	fair	fair	fair	fair	—	excellent	excellent	excellent	excellent	
/2011-06/	poor	poor	fair	fair	fair	fair	good	—	excellent	excellent	excellent	
/2012-06/	poor	poor	fair	fair	fair	fair	good	good	—	excellent	excellent	
/2013-06/	poor	poor	fair	fair	fair	fair	good	good	good	—	excellent	
/2015-01/	poor	poor	fair	fair	fair	fair	good	good	good	good	—	
Downgrades												

Key to quality

- **poor** (very lossy) - the bare minimum of metadata is preserved to allow image display, all other metadata is lost
- **fair** (lossy) - a portion of the metadata is preserved, at least enough to display the image and some other data, it will be far from complete however
- **good** - most information is preserved, it may be possible to do a better job but could be difficult for technical reasons or require custom code not just a transform
- **excellent** - as much information as possible is preserved, some values can still be lost if there are completely incompatible with the new schema

Matrix of transformation paths

This shows the sequence of transformations used to convert one version of the schema to another version.

Source	Targets													
	/2003-FC/	/2003-FC/-TIFF	/2007-06/	/2007-06/ V2	/2008-02/	/2008-02/ V2	/2008-09/	/2009-09/	/2010-04/	/2010-06/	/2011-06/	/2012-06/	/2013-06/	/2015-01/
/2003-FC/	—		xsit: very lossy	xsit: very lossy	via: /2007-06/ very lossy	via: /2007-06/ very lossy	xsit: good	via: /2008-09/ good	via: /2008-09/ & /2009-09/ good	via: /2008-09/ & /2009-09/ & /2010-04/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good
/2003-FC/-TIFF	code	—	xsit: very lossy	xsit: very lossy	via: /2007-06/ very lossy	via: /2007-06/ very lossy	xsit: good	via: /2008-09/ good	via: /2008-09/ & /2009-09/ good	via: /2008-09/ & /2009-09/ & /2010-04/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good
/2007-06/	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	—	—	xsit: very lossy	xsit: very lossy	xsit: good	via: /2008-09/ good	via: /2008-09/ & /2009-09/ good	via: /2008-09/ & /2009-09/ & /2010-04/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	
/2007-06/ V2	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	—	—	xsit: very lossy	xsit: very lossy	xsit: good	via: /2008-09/ good	via: /2008-09/ & /2009-09/ good	via: /2008-09/ & /2009-09/ & /2010-04/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	
/2008-02/	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2003-FC/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2003-FC/ & /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	—	—	xsit: good	via: /2008-09/ good	via: /2008-09/ & /2009-09/ good	via: /2008-09/ & /2009-09/ & /2010-04/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	
/2008-02/ V2	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2003-FC/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2003-FC/ & /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ very lossy	—	—	xsit: good	via: /2008-09/ good	via: /2008-09/ & /2009-09/ good	via: /2008-09/ & /2009-09/ & /2010-04/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	via: /2008-09/ & /2009-09/ & /2010-04/ & /2011-06/ good	
/2008-09/	via: /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2009-09/ & /2010-04/ & /2010-06/ very lossy	via: /2003-FC/ & /2009-09/ & /2010-06/ very lossy	via: /2003-FC/ & /2008-09/ & /2009-09/ & /2010-06/ very lossy	via: /2009-09/ & /2010-04/ & /2010-06/ lossy	via: /2009-09/ & /2010-04/ & /2010-06/ lossy	—	xsit: good	via: /2009-09/ good	via: /2009-09/ & /2010-04/ good	via: /2009-09/ & /2010-04/ & /2011-06/ good	via: /2009-09/ & /2010-04/ & /2011-06/ good	via: /2009-09/ & /2010-04/ & /2011-06/ good	
/2009-09/	via: /2010-04/ & /2010-06/ very lossy	via: /2010-04/ & /2010-06/ very lossy	via: /2003-FC/ & /2010-06/ very lossy	via: /2003-FC/ & /2010-06/ very lossy	via: /2010-04/ & /2010-06/ lossy	via: /2010-04/ & /2010-06/ lossy	via: /2010-04/ & /2010-06/ lossy	—	xsit: good	via: /2010-04/ good	via: /2010-04/ & /2011-06/ good	via: /2010-04/ & /2011-06/ good	via: /2010-04/ & /2011-06/ good	via: /2010-04/ & /2011-06/ good
/2010-04/	via: /2010-06/ very lossy	via: /2010-06/ very lossy	via: /2003-FC/ & /2010-06/ very lossy	via: /2003-FC/ & /2010-06/ very lossy	via: /2010-06/ lossy	via: /2010-06/ lossy	via: /2010-06/ lossy	via: /2010-06/ & /2008-02/ lossy	—	xsit: good	via: /2010-06/ good	via: /2010-06/ & /2011-06/ good	via: /2010-06/ & /2011-06/ good	via: /2010-06/ & /2011-06/ good
/2010-06/	xsit: very lossy	xsit: very lossy	via: /2003-FC/ very lossy	via: /2003-FC/ very lossy	xsit: lossy	xsit: lossy	via: /2008-02/ lossy	via: /2008-02/ & /2008-09/ lossy	via: /2008-02/ & /2008-09/ lossy	—	xsit: good	via: /2011-06/ good	via: /2011-06/ & /2012-06/ good	via: /2011-06/ & /2012-06/ good
/2011-06/	via: /2010-06/ very lossy	via: /2010-06/ very lossy	via: /2010-06/ & /2003-FC/ very lossy	via: /2010-06/ & /2003-FC/ very lossy	via: /2010-06/ lossy	via: /2010-06/ lossy	via: /2010-06/ & /2008-02/ lossy	via: /2010-06/ & /2008-02/ & /2008-09/ lossy	via: /2010-06/ & /2008-02/ & /2008-09/ lossy	xsit: good	—	xsit: good	via: /2012-06/ good	via: /2012-06/ & /2013-06/ good
/2012-06/	via: /2011-06/ & /2010-06/ very lossy	via: /2011-06/ & /2010-06/ very lossy	via: /2011-06/ & /2010-06/ & /2003-FC/ very lossy	via: /2011-06/ & /2010-06/ & /2003-FC/ very lossy	via: /2011-06/ & /2010-06/ lossy	via: /2011-06/ & /2010-06/ lossy	via: /2011-06/ & /2010-06/ & /2008-02/ lossy	via: /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/ lossy	via: /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/ lossy	via: /2011-06/ good	xsit: good	—	xsit: good	
/2013-06/	via: /2012-06/ & /2011-06/ & /2010-06/ very lossy	via: /2012-06/ & /2011-06/ & /2010-06/ very lossy	via: /2012-06/ & /2011-06/ & /2010-06/ & /2003-FC/ very lossy	via: /2012-06/ & /2011-06/ & /2010-06/ & /2003-FC/ very lossy	via: /2012-06/ & /2011-06/ & /2010-06/ lossy	via: /2012-06/ & /2011-06/ & /2010-06/ lossy	via: /2012-06/ & /2011-06/ & /2010-06/ & /2008-02/ lossy	via: /2012-06/ & /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/ lossy	via: /2012-06/ & /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/ lossy	via: /2012-06/ good	xsit: good	—		
/2015-01/	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ very lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ very lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ & /2003-FC/ very lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ & /2003-FC/ very lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ & /2008-02/ lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/ lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/ lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/ lossy	via: /2013-06/ & /2012-06/ & /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/ lossy	xsit: good - units lost		

3.8.7 Folders in the OMERO model

OMERO offers both Projects that contain only Datasets and Datasets that contain only Images. Images may thus be organized within a two-level container hierarchy.

Reflecting the [June 2016 release](#) of the [OME Data Model](#), OMERO 5.3's object model adds a new kind of container, the Folder. In many respects they are rather like Datasets: for example, Folders have a description, they may be annotated and they may contain Images. However, they are different from Datasets in important respects.

Folders may contain:

- Images
- Regions of Interest (ROIs)
- other Folders

or a heterogeneous mix of the above.

For organizing data one may use Folder hierarchies of arbitrary depth. Just as an Image may be in multiple Datasets at once, the same Folder may be in multiple Folders at once. However, there is an acyclicity constraint: a Folder may *not* contain itself, even indirectly.

The OMERO graphical clients offer very limited support for Folders. At present Folders may be most useful for those working with their data via the *OMERO Application Programming Interface* and its gateways or with the *OMERO.cli obj plugin*. The *measurement tool* in OMERO.insight shows the Folders that ROIs are in. OMERO.web is yet to provide support for Folders.

3.9 Searching

3.9.1 OMERO search

OMERO.server uses [Lucene](#) to index all string and timestamp information in the database, as well as all `OriginalFile` which can be parsed to simple text (see [File parsers](#) for more information). The index is stored under `/OMERO/FullText` or the `FullText` subdirectory of your `omero.data.dir`, and can be searched with Google-like queries.

Once an entity is indexed, it is possible to start writing querying against the server via `IQuery.findAllByFullText()`. Use `new Parameters(new Filter().owner())` and `.group()` to restrict your search. Or alternatively use the `ome.api.Search` interface (below).

See also:

[Search and indexing configuration](#)

Section of the sysadmin documentation describing the configuration of the search and indexing for the server.

Field names

Each row in the database becomes a single Lucene Document parsed into the several Fields. A field is referenced by prefixing a search term with the field name followed by a colon. For example, *name:myImage* searches for myImage anywhere in the name field.

Field	Comments
	Any unprefixed field searches the combination of all fields together i.e. a search for <i>cell AND name:myImage</i> gets translated to <i>combined_fields:cell AND name:myImage</i> .
<field name>	Each string, timestamp, or <code>Details</code> field of the entity also gets its own Field entry, like the name field above
de-tails.owner.omeName	Login name of the owner of the object
de-tails.owner.firstName	First name of the owner of the object
de-tails.owner.lastName	Last name of the owner of the object
details.group.name	Group name of the owning group of the object
de-tails.creationEvent.id	Id of the Event of this objects creation
de-tails.creationEvent.tin	When that Event took place
de-tails.updateEvent.id	Id of the Event of this objects last modification

continues on next page

Table 2 – continued from previous page

Field	Comments
de-tails.updateEvent.time	When that Event took place
details.permissions	Permissions in the form <i>rwrwrw</i> or <i>rw-</i>
tag	Contents from a <code>TagAnnotation</code> .
annotation	Contents from annotations, including <code>TagAnnotation</code> and any <code>TextAnnotation</code> on another <code>TextAnnotation</code> (a.k.a. a <i>description</i>). Non-string annotations like <code>FileAnnotation</code> are not covered by this definition and are handled separately. See below.
annotation.ns	Namespace (if present) for any annotations on an object
annotation.type	Short type name, e.g. <code>TextAnnotation</code> or <code>FileAnnotation</code> for any annotations on an object
channel.name	Name of the <code>Channel</code> object
channel.fluor	Fluor value of the <code>Channel</code> object (e.g. “Alex Fluor 488” or “DAPI”)
channel.mode	Mode of the <code>Channel</code> object (e.g. “BrightField” or “SPIM”)
channel.photometricInterp	Name of the <code>Channel</code> object (e.g. “RGB” or “Monochrome”)
file.name	For <code>FileAnnotation</code> and objects they are attached to, the name of the <code>OriginalFile</code>
file.format	For <code>FileAnnotation</code> and objects they are attached to, the format of the <code>OriginalFile</code>
file.path	For <code>FileAnnotation</code> and objects they are attached to, the path of the <code>OriginalFile</code>
file.shal	For <code>FileAnnotation</code> and objects they are attached to, the shal of the <code>OriginalFile</code>
file.contents	For <code>FileAnnotation</code> and objects they are attached to as well as the <code>OriginalFile</code> itself, the file contents themselves if their Format is configured with the File parsers.
fileset.entry.name	Name of an imported file.
file-set.entry.clientPath	Original, client-side path of an imported file.
file-set.templatePrefix	Location of the import in the managed repository.
<code>\${NAME}</code>	For <code>MapAnnotation</code> and objects they are attached to, dynamic fields are generated for each of the <code>NamedValue</code> entries in the annotation. For example, if <code>NamedValue('temperature', '37')</code> is one such value, a field named <code>temperature</code> will exist.
has_key	As <code>\${NAME}</code> , but a single field of name <code>has_key</code> is generated for each <code>NamedValue</code> entry with a value of the key such that a search for <code>has_key:temperature</code> in the example above is possible.
Internal	
combined_fields	The default field prefix.
_hibernate_class	Used by Hibernate Search to record the entity type. The class value, e.g. <code>ome.model.core.Image</code> is also entered in <code>combined_fields</code> . Unimportant for the casual users.
id	The primary key of the entity. Unimportant for the casual user

Queries

Search queries are very similar to Google searches. When search terms are entered without a prefix (“name:”), then the default field will be used which combines all available fields. Otherwise, a prefix can be added to restrict the search. The search term is first split into “tokens” and these are combined into a search query. The tokenizing happens on all non-alpha-numerical characters, such as space, underscore, hyphen etc. The query is built by combining the tokens with an “OR” operator (see examples in the “Indexing” paragraph). The search terms or the tokens created from them as above must precisely match the indexed entries. This means for example that a search term *tes* will **not** match the indexed entry *test* and the search will accordingly give no result.

Indexing

Successful searching depends on understanding how the text is indexed. The default analyzer used is [the FullTextAnalyzer](#).

1. Desktop/image_GFP-H2B_1.dv	--->	"desktop", "image", "gfp", "h2b", "1", "dv"
2. Desktop/image_GFP-H2B_2.dv	--->	"desktop", "image", "gfp", "h2b", "2", "dv"
3. Desktop/image_GFP_01-H2B.dv	--->	"desktop", "image", "gfp", "01", "h2b", "dv"
4. Desktop/image_GFP-CSFV_a.dv	--->	"desktop", "image", "gfp", "csfv", "a", "dv"

Assuming these entries above for Image.name:

- searching for **GFP-H2B** returns 1, 2, 3 and 4, because of the tokenizing on the hyphen and joining of the tokens by an **OR**.
- searching for **"GFP H2B"** or **"GFP-H2B"** only returns 1 and 2, since the quotes enforce the exact sequence of the tokens and the query is built with an **AND**.
- searching for **GFP H2B** returns 1, 2, 3 and 4, since the two tokens are joined by an **OR**.

With the same entries as above and adding a wildcard:

- searching for ***FP** returns 1, 2, 3 and 4. As this example shows, **leading wildcards in the Graphical User Interface are allowed**, but must be explicitly enabled when using the API directly, see below in the developers section.
- searching for **GF*** returns 1, 2, 3 and 4.
- searching for **GFP.*** returns no results, but **GFP.*** returns 1, 2, 3 and 4. Only hyphen and underscore do not return results in this situation, the other non-alpha-numerical characters do.

Wildcards and quotations:

Wildcard inside quotations is not parsed as a wildcard, but as a non-alpha-numerical character on which the tokenizing happens.

- searching for **"*FP-H2B"** returns no results, since it is the same as searching for **"FP-H2B"**.
- searching for **"GF*"** returns no results, since it is the same as searching for **"GF"**.
- searching for **"GFP.*"** returns 1, 2, 3 and 4, since it is the same as searching for **"GFP."**.
- searching for **"GFP*H2B"** returns 1 and 2, since it is the same as searching for **"GFP H2B"**.

Information for developers

ome.api.IQuery

The current IQuery implementation restricts searches to a single class at a time.

- `findAllByFullText(Image.class, "metaphase")` – Images which contain or are annotated with "metaphase"
- `findAllByFullText(Image.class, "annotation:metaphase")` – Images which are annotated with "metaphase"
- `findAllByFullText(Image.class, "tag:metaphase")` – Images which are tagged with "metaphase" (specialization of the previous)
- `findAllByFullText(Image.class, "file.contents:metaphase")` – Images which have files attached containing "metaphase"

- `findAllByFullText(OriginalFile.class, "file.contents:metaphase")` – File containing “metaphase”

ome.api.Search

The Search API offers a number of different queries along with various filters and settings which are all maintained on the server.

The matrix below show which combinations of parameters and queries are supported (S), will throw an exception (X), and which will simply be ignored (I).

Query Method →	byGroupForTags/byTagsForGroup	byFull-Text/SomeMustNone	byAnnotated-With
Parameters			
annotated between	S	S	S
annotated by	S	S	S
annotated by	S	I	I
created between	S	I	I
modified between	S	I (Immutable)	S
owned by	S	S	S
all types	X	I	X
1 type	S	I	S
N types	X	I	X
only ids	S	I	S
Ordering / Fetches			
orderBy	S	I	S
fetchAnnotations	¹	I	²
Other			
setProjections ³	X	X	X
current*Metadata ⁴	X	X	X

Leading wildcard searches

Leading wildcard searches are disallowed by default. “?omething” or “*hatever”, for example, would both throw exceptions. They can be run by using:

```
Search search = serviceFactory.createSearchService();
search.setAllowLeadingWildcards(true);
```

There is a performance penalty, however. In addition, wildcard searches get expanded on the server to boolean queries. For example, assuming “ACELL”, “BCELL”, and “CCELL” are all terms in your index, then the query:

```
*CELL
```

gets expanded to:

¹ Any fetchAnnotation() argument to byFullText() or related queries, returns **all** annotations.

² byAnnotatedWith() does not accept a fetchAnnotation() argument of **Annotation.class**.

³ setProjects may need to be removed if Lucene cannot handle OMERO’s security requirements.

⁴ Not yet implemented.

ACELL OR BCELL OR CCELL

If there are too many terms in the expansion then an exception will be thrown. This requires the user to enter a more refined search, but not because there are too many results, only because there is not enough room in memory to search on all terms at once.

Extension points

Two extension points are currently available for searching. The first are the *File parsers* mentioned above. By configuring the map of Formats (roughly mime-types) of files to parser instances, extracting information from attached binary files can be made quick and straightforward.

Similarly, *Search bridges* provide a mechanism for parsing all metadata entering the system. One built in bridge (the `FullTextBridge`) parses out the fields mentioned above, but by creating your own bridge it is possible to extract more information specific to your site.

See also:

Working with annotations, *Search bridges*, *File parsers*, *Query Parser Syntax*,

Luke

a Java application which you can download and point at your `/OMERO/FullText` directory to get a better feeling for Lucene queries.

3.9.2 File parsers

File parsers extract text from various file types and provide it as a `Reader` to the `FullTextBridge` for use during search indexing. Plain text formats can use the default `fileParser` bean, but any specialized format, such as PDF or RTF requires special libraries and special registration.

Configuration

Currently, configuration takes places solely in `service-ome.api.Search.xml`. Eventually, it should be able to replace file parsers at configuration or even runtime.

Available parsers

File type	Parser
application/pdf	https://pdfbox.apache.org
text/xml	(internal)
text/plain	(internal)
text/csv	(internal)

The base class for File parsers are `FileParser.java`

See also:

OMERO search

3.9.3 Search bridges

Warning: All search bridge classes have been deprecated in 5.3.0. This extension has never been widely used and is now out of date, causing a bottleneck in performance and preventing the upgrade of Hibernate. A new modern search functionality is under development.

A “bridge” is the mapping between your metadata and how it is stored in the `Lucene` <<https://lucene.apache.org>>`\_index. *OMERO search* uses one internal bridge to parse all of your metadata for later searching. If, however, there is more metadata that you would like to add to the index, you can implement the `org.hibernate.search.bridge.FieldBridge` interface yourself, or subclass the helper class <https://github.com/ome/omero-server/blob/v5.6.10/src/main/java/ome/services/fulltext/BridgeHelper.java>

Example

Assume you wanted to be able to search for a project based on the name of all images contained in that project. In the `set` method,

```
public void set(final String name, final Object value,
               final Document document, final Field.Store store,
               final Field.Index index, final Float boost) {
```

you would need to add a field to the Document for each Image.

```
Project p = (Project) value;
List<Image> images = getImages(p);
for (Image image : images) {
    add(document, "image_name", image.getName(), store, index, boost);
}
```

Configuration

Custom bridges are configured in *Search* but can be overridden via the *standard configuration mechanisms*. The `omero.search.bridges` property defines a comma-separated list of bridge classes which will be passed to <https://github.com/ome/omero-server/blob/v5.6.10/src/main/java/ome/services/fulltext/FullTextBridge.java>.

See *Java deployment* for how to have your bridge classes included on the server’s classpath if it doesn’t get built by the *Build System*.

Available bridges

See <https://github.com/ome/omero-server/tree/v5.6.10/src/main/java/ome/services/fulltext/bridges> for a list of provided (example) bridges.

Re-indexing

BridgeHelper provides two methods – `reindex(IObject)` and `reindexAll(List<IObject>)` – for keeping the indexes for objects in sync.

For example, if the `image.name` above were to be changed, the index for the `Project` would be stale until the `Project` itself were re-indexed. Custom bridges can call `reindex(Project)` while indexing the image to have the `Project` re-indexed from the **backlog**. Before any new changes are processed, the backlog is always first cleared. One caveat: while processing the backlog, no new objects can be added to it.

For bridge writers, this means concretely that implementations should check for all related types and index them in groups, rather than relying on transitivity. For example,

```
if (value instanceof Project) {
    final Project p = (Project) value;
    handleProject(p, document, store, index, boost);

    for (final ProjectDatasetLink pdl : p.unmodifiableDatasetLinks()) {
        final Dataset d = pdl.child();
        reindex.add(d);
        handleDataset(d, document, store, index, boost);

        for (final DatasetImageLink dil : d.unmodifiableImageLinks()) {
            final Image i = dil.child();
            reindex.add(i);
            handleImage(document, store, index, two_step_boost, i);
        }
    }
} else if (value instanceof Dataset) {
    final Dataset d = (Dataset) value;
    handleDataset(d, document, store, index, boost);

    for (final ProjectDatasetLink pdl : d.unmodifiableProjectLinks()) {
        final Project p = pdl.parent();
        reindex.add(p);
        handleProject(p, document, store, index, two_step_boost);
    }

    for (final DatasetImageLink dil : d.unmodifiableImageLinks()) {
        final Image i = dil.child();
        reindex.add(i);
        handleImage(document, store, index, two_step_boost, i);
    }
} else if (value instanceof Image) {
    final Image i = (Image) value;
    handleImage(document, store, index, two_step_boost, i);

    for (final DatasetImageLink dil : i.unmodifiableDatasetLinks()) {
        final Dataset d = dil.parent();
        reindex.add(d);
        handleDataset(d, document, store, index, boost);
    }
}
```

(continues on next page)

(continued from previous page)

```

        for (final ProjectDatasetLink pdl : d
            .unmodifiableProjectLinks()) {
            final Project p = pdl.parent();
            reindex.add(p);
            handleProject(p, document, store, index, boost);
        }
    }
}

//
// Handle re-indexing
//
if (reindex.size() > 0) {
    reindexAll(reindex);
}
}

```

In which case, regardless of whether an Image, Dataset, or Project is indexed, all related objects are simultaneously added to the backlog, which will be processed in the next cycle, but **their** indexing will not add any new values to the backlog.

See [#955](#) and [#1102](#)

See also:

OMERO search

3.10 Authentication and Security

3.10.1 Password Provider

A *Password Provider* is an implementation of the Java interface `ome.security.auth.PasswordProvider`. Several implementations exist currently:

- `ome.security.auth.JdbcPasswordProvider` is the most common provider, and uses the “password” table for storing passwords hashed using MD5 and salt per user.
- `ome.security.auth.FilePasswordProvider` is rarely used, but in some scenarios may be useful since it permits setting usernames and passwords in a plain text file.
- `ome.security.auth.LdapPasswordProvider` is a highly configurable provider which provides READ-ONLY access to an LDAP server and can create users and groups on the fly. See *LDAP plugin design* for more information.

The `omero.security.password_provider` property (see *Security*) defines the implementation of *PasswordProvider* that will be used to authenticate users. Chains of password providers can be created using `ome.security.auth.PasswordProviders`. For instance, the default server authentication uses *chainedPasswordProvider* which first checks the `LdapPasswordProvider` and then falls back to the `JdbcPasswordProvider`.

To write your own provider, you can either subclass from `ome.security.auth.ConfigurablePasswordProvider` as the providers above do, or write your own implementation from scratch. You will need to define your object and optionally your new chained password providers in a Spring XML file matching the pattern `ome/services/db-*.xml`. See *Extending OMERO.server* more for information.

Things to keep in mind

- All the existing implementations take care to publish a `LoginAttemptMessage` so that any `LoginAttemptListener` implementation can properly react to failed logins. Your implementation should probably do the same.
- When dealing with chains of password providers, an implementation can safely return null from `checkPassword` to say “I don’t know anything about this”. This is only important if you configure your own chained password provider with your new implementation as one of the elements.
- Due to the service dependency order, new password providers defined in the Spring XML file should be configured with lazy initialization (`lazy-init="true"`) so that the beans are only created when needed.

3.10.2 LoginAttemptListener

All the `Password Provider` implementations provided by default publish a “`LoginAttemptMessage`” every time they check a password value. This permits any `org.springframework.context.ApplicationListener<LoginAttemptMessage>` to react to the login. Only one implementation is active by default (as of 4.2.1): `ome.security.auth.LoginAttemptListener` which throttles logins after a given number of failed attempts. Configuration for this listener is available in *Security*:

```
omero.security.login_failure_throttle_count=1 # Number of failed attempts before
↪ throttling begins
omero.security.login_failure_throttle_time=3000 # Time in milliseconds
```

A more sophisticated listener would lock the user’s account until an administrator intervenes. This is the goal of #3139.

3.10.3 LDAP plugin design

Once configured, *LDAP authentication* allows sysadmins to control OMERO’s user and group creation via an external, locally-maintained LDAP server. Due to the flexibility of LDAP, each instance may have a number of requirements that cannot be supported out of the box. Below, we discuss the design of the LDAP plugin as well as how it can be extended for local use.

Simple walkthrough

The LDAP plugin follows these steps:

1. Sysadmin configures properties mapping users and groups from LDAP to OMERO.
2. Once LDAP is enabled, any OMERO user who has a value of `true` in the `ldap` column of the `experimenter` table will have their password checked against LDAP and **not** against OMERO (changing the password via OMERO is *not* supported). This functionality is provided by the *Password Provider*. The DN (Distinguished Name) is not stored in the OMERO DB and is retrieved from the LDAP server on each user login.
3. If there is no OMERO user for a given name, the LDAP plugin will use `omero.ldap.user_filter` and `omero.ldap.user_mapping` to look for a valid user:
 1. The `user_mapping` property is of the form: `omeName=<ldap attribute>; firstName=<ldapAttribute>;...`
 2. For looking up new users, the plugin will only use the `omeName` attribute. For example, if a user tries to login with “emma” and the `user_mapping` starts with `omeName=cn`; then the LDAP search will be for `(cn=emma)`.

3. The (cn=emma) LDAP filter is then added to the value of `omero.ldap.user_filter`. For example, if the user filter is (objectClass=inetOrgPerson), the full query for the new user will be: (&(objectClass=inetOrgPerson)(cn=emma))
4. If the search returns a **single** LDAP user, then an OMERO user will be created with all properties mapped according to `omero.ldap.user_mapping` and the `ldap` property set to `true`.
5. Then the user will be placed in groups according to the value of `omero.ldap.new_user_group`, which are created if necessary. Details of the various options can be found under [LDAP authentication](#). Each option is handled by a different `NewUserGroupBean` implementation.

NewUserGroupBean.java

The interface described for the “:bean:” `new_user_group` prefix, is `ome.security.auth.NewUserGroupBean`. It defines a single method: `groups(..., AttributeSet set)` which returns a list of `ExperimenterGroup` ids (`List<Long>`) which the user should be added to.

Other prefix handlers also implement the interface as examples. In the same package are:

- **:attribute:** -
 [AttributeNewUserGroupBean.java](#)
- **:ou:** -
 [OrgUnitNewUserGroupBean](#)
- **:query:** -
 [QueryNewUserGroupBean](#)

See also:

OMERO.server installation

Instructions for installing OMERO.server on UNIX and UNIX-like platforms

Server security and firewalls

General instructions on server security

3.10.4 OMERO roles

There are two areas where roles are used. The first is in service-level security (**deciding who can make what calls**) and the second is in object-level security (**who can read and edit individual objects**). Both of these sets of roles are composed of “`ExperimenterGroups`”.

Setting roles

An Experimenter is given a role by being a member of an `ExperimenterGroup` (specifically, this means that there exists a `GroupExperimenterMap` where `child == the experimenter id` and `parent == the experimenter group id`). Creating a `GroupExperimenterMap` is generally done transparently by `IAdmin` service. Instead, administrators call:

- `IAdmin.createUser(user)`
- `IAdmin.createGroup(group)`
- `IAdmin.addGroups(user, group, group, ...)`
- `IAdmin.removeGroups(user, group, group, ...)`
- `IAdmin.createSystemUser(user)`

Service-level

The two main roles that are distinguished at the service-level are “system” and “user” groups. These groups are created during installation and must not be configured by administrators. All users added through `IAdmin.createUser(user)` are automatically added to the “user” group, and all users added through `IAdmin.createSystemUser(user)` are added to both “system” and “user” groups.

During login, a user is checked against all groups for membership in “user” or “system”, and no special action needs to be taken by the user or client developer.

Note: Although currently all methods in the session beans are labelled as `@RolesAllowed("user")` or `@RolesAllowed("system")`, there is nothing stopping a developer from writing a service method which accepts another role, as long as that role has been created in the `ExperimenterGroup` table.

Object-level

Object-level security is more complicated. When execution reaches the `EventHandler`, a second login takes place to authorize the user with the OMERO security system. This second authorization process takes into account the group that (can be) passed into the client `ServiceFactory\ (Login)` via `Login(String,String,String,String)`. If a user has not set the group name or the default “user” group has been set, then the default group for that user will be used (a user is not allowed to use the “user” group for object updates). If the group is set to “system”, then the “system” group **will** be used, and a user is granted admin privileges for object updates. This means that a user could be authorized to call a method by being in the “system” group, but if the “system” group is not specified, `SecurityViolations` will most likely be thrown.

Special privileges for PIs

There is one other special, implicit role which is group leader. The user listed as “owner” for a group is considered the group leader, also known as the PI (principal investigator) of that group. For all objects that are assigned to that group, the PI has near-admin access. Objects which are set to unreadable (“-wu-wu-wu”) will still be visible to the PI. The same objects can also be updated regardless of the permissions set.

3.10.5 OMERO security system

The OMERO security system is intended to be as transparent as possible while permitting users to configure the visibility of their data. For the user, this means that with no special actions, data and metadata created will be readable by both members of the same group and by other users, but will be writable by no one, comparable to a umask of 755 on Unix. For the developer, transparency means that there is little or no code that must be written to prevent security violations, and simple mechanisms for allowing restricted operations when the time comes.

Other links which may be of use:

- [*OMERO admin interface*](#)
- [*OMERO roles*](#)
- [*Groups and permissions system*](#)
- [*OMERO permissions querying, usage and history*](#)

Concepts

Several concepts and/or components from our and other code bases play a role in OMERO security:

Hibernate Listeners and Events

listeners and events are the two extension points provided by Hibernate for responding to and influencing internal actions. Essentially any method on the `org.hibernate.Session` interface has a corresponding event, and almost the same is true for the interceptor. Additionally interceptors can change the state of the objects before INSERT and UPDATE, and after SELECT.

Hibernate Filters

filters are a mechanism for injecting SQL clauses into the SELECT statements generated by Hibernate. Similar to listeners and events for write actions, filters allow us to extend Hibernate functionality with our own logic.

Handler/interceptor

as outlined in *Aspect-oriented programming*, OMERO makes extensive use of method interceptors to relieve the developer of some coding burden. Transactions, session management, and, naturally, security are handled largely by our interceptors (or “handlers”).

Events

Every write action produces an Event in the database. This database contains several EventLogs which specify exactly what was created or altered during that specific event.

Participants

Now, with the concepts cleared up, we can take a look at all of the concrete source artifacts (“participants”) which are important for security.

Top-level and build

omero-model.properties

contains login and connection information for the database.

build.properties.example

contains the default root password. This can be overridden with your own `etc/local.properties` file.

hibernate.properties

contains default connection information for the database, this includes the user name and if necessary the user password. These values can be overridden in `local.properties`.

omero.properties

contains a default user group, event type, and connection information for logging in from the client side, if no Login or Server is specified to ServiceFactory. These values can be overridden in `local.properties`.

object.vm

specifies the default permissions that all objects will have after construction, as well as attaches the security filter to all classes and collections.

psql-footer.vm

used by DSLTask to generate `psql-footer.sql` which is used to bootstrap the database security system (root et al).

Client and common

The server uses the information in `etc/local.properties` to create a Login object. If no Login, Server, or Properties is provided to the ServiceFactory constructor, the empty properties defined in `ome/config.xml` are used.

IAdmin.java

main interface for administering accounts and privileges. See *OMERO admin interface* for more.

ITypes.java

only related to security by necessity. The security system disallows the creation of certain “System-Types”. Enumerations are one of these. ITypes, however, provides a createEnumeration method with general access.

GraphHolder.java

all model objects (implementations of IObject have a never-null GraphHolder instance available. This graph holder is responsible for various OMERO and Hibernate internal processes. One of these is the exchange of Tokens. For the server, the existence of a special token within the GraphHolder grants certain privileges to that IObject. This logic is encapsulated within the SecuritySystem.

Details.java

contains all the fields necessary to perform access control, such as owner, group, and permissions.

Permissions.java

representation of rights and roles. For more information, see *Groups and permissions system*.

Token.java

an extremely simple class (“public class Token { }”) which is only significant when it is equivalent (“==”) to a privileged Token stored within the SecuritySystem.

IEnum.java

the only non-access control related types which are considered “System-Types” are enumerations. IEnum is a marker interface for all enumerations and creation of IEnum implementations can only be performed through ITypes.

SecurityViolation.java

the exception thrown by the *OMERO security system* at the first hint of misdoings.

Principal.java

an Omero-specific implementation of the java.security.Principal interface. Carries in addition to the typical name field, information about the user group, the event type, and the session umasks.

`meta.ome.xml`

JBoss-only

`ServiceFactory.java Login.java Server.java`

Server side

`AdminImpl.java CurrentDetails.java SecureAction.java SecuritySystem.java BasicSecuritySystem.java ACLEventListener.java EventHandler.java MergeEventListener.java OmeroInterceptor.java SessionHandler.java SecurityFilter.java EventLogListener.java EventListenersFactoryBean.java LocalAdmin.java hibernate.xml sec-system.xml services.xml`

End-to-end

Build system

Security starts with the build system and installation. During the generation of the model (by the DSLTask), a sql script is created called “data.sql”. After ddl.sql creates the database, data.sql bootstraps the security system by creating the initial (root) experimenter, and event, and then creates the “system” group and the “user” group. It then creates a password table and sets the root password to “ome”. (It also creates all of the enumeration values, but that is unimportant for security).

Note: The password table is not mapped into Hibernate, and is only accessible via the *OMERO admin interface*.

Client-side

To begin the runtime security process, a user logs in by providing a Login and/or a Server instance to ServiceFactory. These types are immutable and their values remain constant for the lifetime of the ServiceFactory. The user can also set the umask property on ServiceFactory_. This value is mutable and can be set at anytime.

The values are converted to *java.util.Properties* which are merged with the properties from the *.properties files to create the client *OmeroContext* (also known as the “application context”). The context contains a Principal and user credentials (password, etc.) which are associated with the thread before each method execution in a specialized TargetSource. Finally, these objects are serialized to the application server along with the method arguments.

Application server

The application server first performs one query (most likely SQL) to check that the credentials match those for the given user name. A second query is executed to retrieve all roles/groups for the given user. If the roles returned are allowed to invoke the desired method, invocation continues with the queried user and roles stored in the InvocationContext.

Server code

Execution then passes to OMERO code, specifically to the interceptors and lifecycle methods defined on our session beans. This intercepting code checks the passed Principal for OMERO-specific information. If this information is available, it is passed into the SecuritySystem through the login method. Finally, execution is returned to the actual bean which can either delegate to OMERO services or perform logic themselves.

Interceptors

All calls to the delegates (and in the future all calls on the session beans) are also caught intercepted by Spring-configured interceptors. These guarantee that the system is always in a valid and secure state. In stack order they are:

- the service handler, which handles logging and checks all arguments against ServiceInterface annotations;
- the proxy handler, which after execution, removes all uninitialized Hibernate objects to prevent exceptions (special logic allows this to happen See unloaded objects);
- the transaction handler, which binds a transaction to the thread,
- the session handler, which uses the now prepared transaction to initialize either a new or a cached (in the case of stateful session beans) session and also bind it to the thread;

- and finally, the event handler, which performs what one might actually consider login. It instantiates `Experimenter`, `ExperimenterGroup`, and `Event` objects from Hibernate and gives them a special `Token` so that they can authenticate themselves later to the `SecuritySystem` and turns session read security on for the entirety of execution below its frame.

Services

Finally execution has reached the OMERO services and can begin to perform logic. Because of these layers, almost no special logic (other than eviction and not calling write methods from within read methods. see #223) needs to be considered. There are, however, a few special cases.

IQuery (within the application server), for example will always return a graph of active Hibernate objects. Changes to them will be persisted to the database on flush.

IUpdate, on the other hand, does contain some logic for easing persistence, though this will eventually be ported to the Hibernate event system. This includes pre-saving the newly created event and the work of `UpdateFilter` like reloading objects unloaded by the proxy handler (above).

Finally, **IAdmin** is special in that it and it alone access the non-Hibernate password data store and even access application server APIs (like JMX) in order to make authentication and authorization function properly.

Hibernate

Once execution has left this service layer, it enters the world of Hibernate ORM. Here we cannot actively change functionality but only provide callbacks like the `OmeroInterceptor` and `EventListeners`. The `OmeroInterceptor` instance registered with the Hibernate `SessionFactory` (via Spring) is allowed for calling back to the often mentioned `SecuritySystem` to determine what objects can be saved and which deleted. It also properly sets the, for a user mostly unimportant, `Details` object. The `EventListeners` are more comprehensive than the `OmeroInterceptor` and can influence almost every phase of the Hibernate lifecycle, specifically every method on the `Session` interface.

The event listeners which implement `AbstractSaveEventListener` (i.e. `MergeEventListener`, `SaveOrUpdateEventListener`, etc.) are responsible for reloading unloaded objects (and will hopefully take this functionality fully from `IUpdate`) and provide special handling for enums and other system types. There are also event listeners which are the equivalent of database triggers (pre-update, post-delete, etc.) and these are used for generating our audit log.

So much for write activities. Select queries are, as mentioned above, secured through the use of Hibernate filters which add join and where clauses dynamically to queries. For example an HQL query of the form:

```
select i from Image i
```

would be filtered so that the current user does not receive references to any objects with reduced visibility:

```
select i from Image i where ( current_user = :root OR i.permissions = :readable )
```

The actual clauses added are much more complex and are added for each joined entity type (i.e. table) which appears in a query.

```
select i from Image i join i.defaultPixels p
```

would contain the “(current_user = :root ...)” clause twice.

Currently, subqueries are an issue in that the clauses do not get added to them. This may cause consternation for some particular queries.

Security system

All of this is supported by an implementation of the `SecuritySystem` interface which encapsulates all logic regarding security. It also hides as much as it can, and if not specifically needed should be ignored. However, before you attempt to manually check security, by all means use the security system, and for that, it may need to be acquired from the server-side *OmeroContext*. Currently, there is no client-side security system. See #234.

The *OMERO security system* and its current only implementation `BasicSecuritySystem` are somewhat inert and expect well-defined and trusted (see #235) methods to invoke callbacks during the proper Hibernate phase.

Logging in (client-side)

When using the client library and the `ServiceFactory`, logging in is trivial. One need only set several System properties or place them in an `omero.properties` file somewhere on the classpath. Internally, Spring takes the System properties and creates an `ome.system.Principal` instance. This is then passed to the server on each invocation of a proxy obtained from JNDI.

Logging in (server-side)

Much of this infrastructure is not available to server-side code (no `ome/client/spring.xml`, no `ServiceFactory`, etc.). As such, the `Principal` needs to be manually created and provided to the server-side `SecuritySystem.java`.

Basically it amounts to this:

```
Principal p = new Principal( omeroUserName, omeroGroupName, omeroEventTypeValue );
securitySystem.login( p );
```

This must be run otherwise the `EventHandler` will throw a security exception.

Note: The code above is being run in a secure context (i.e. you are root). Please be careful.

3.10.6 OMERO permissions querying, usage and history

Working with the OMERO 5.6.10 permissions system

Example environment

- OMERO 5.6.10 server
- IPython shell initiated by running `omero shell --login`

Group membership

User	private-1	read-only-1	read-write-1	read-annotate-1
user-2	Yes	Yes	No	No
user-3	No	Yes	No	Yes

Simple inserts and queries

While the ‘Default Group’ is essentially a deprecated concept, a user must be logged into one to provide a default context. It is still possible to change this default group but it is no longer required to make queries in other permissions contexts.

All remote calls to an OMERO server, since well before version 4.1.x, have the option of taking an Ice context object. Through this object, and manipulations thereof, we can affect our query context. What follows is a series of examples exploring inserts and queries using contexts that span a single group at a time.

Retrieving a user’s event context and group membership

```
#!/python
# Session that has already been created for user-2
session = client.getSession()

# Retrieve the services we are going to use
admin_service = session.getAdminService()

ec = admin_service.getEventContext()
print(ec)
groups = [admin_service.getGroup(v) for v in ec.memberOfGroups]
for group in groups:
    print('Group name: %s' % group.name.val)
```

Example output:

```
object #0 (::omero::sys::EventContext)
{
    shareId = -1
    sessionId = 1783
    sessionId = 213adc46-2c5f-449b-81fc-fe24dec38b58
    userId = 10
    userName = user-2
    groupId = 9
    groupName = private-1
    isAdmin = False
    eventId = -1
    eventType = User
    memberOfGroups =
    {
        [0] = 9
        [1] = 8
    }
```

(continues on next page)

(continued from previous page)

```

    [2] = 1
  }
  leaderOfGroups =
  {
  }
  groupPermissions = object #1 (::omero::model::Permissions)
  {
    _restrictions =
    {
    }
    _perm1 = -120
  }
}

Group name: private-1
Group name: read-only-1
Group name: user

```

Here you can see and validate that, when logged in as user-2, we are a member of both the `private-1` and `read-only-1` groups. Membership of the `user` group is required in order to login. This group essentially acts as a role, letting the OMERO security system know whether or not the user is active.

Inserting and querying data from specific groups

For the purposes of this example, we will prepare a single Project in both the `private-1` and `read-only-1` groups and then perform various queries on those Projects.

```

#!/python
from omero.model import *
from omero.rtypes import *
from omero.sys import ParametersI
from omero.cmd import Delete
from omero.callbacks import CmdCallbackI

# Session that has already been created for user-2
session = client.getSession()

# Project object instantiation
private_project = ProjectI()
private_project.name = rstring('private-1 project')
read_only_project = ProjectI()
read_only_project.name = rstring('read-only-1 project')

# Retrieve the services we are going to use
update_service = session.getUpdateService()
admin_service = session.getAdminService()
query_service = session.getQueryService()

# Groups we are going to write data into
private_group = admin_service.lookupGroup('private-1')
read_only_group = admin_service.lookupGroup('read-only-1')

```

(continues on next page)

(continued from previous page)

```

# Save and return our two projects, setting the context correctly for each
ctx = {'omero.group': str(private_group.id.val)}
private_project = update_service.saveAndReturnObject(private_project, ctx)
ctx = {'omero.group': str(read_only_group.id.val)}
read_only_project = update_service.saveAndReturnObject(read_only_project, ctx)

private_project_id = private_project.id.val
read_only_project_id = read_only_project.id.val
print('Created Project:%d in group private-1' % (private_project_id))
print('Created Project:%d in group read-only-1' % (read_only_project_id))

# Query for the private project we created using private-1
#
# You will notice that this returns the Project as we have specified
# the group that the Project is in within the context passed to the
# query service.
ctx = {'omero.group': str(private_group.id.val)}
params = ParametersI()
params.addId(private_project_id)
projects = query_service.findAllByQuery(
    'select p from Project as p ' \
    'where p.id = :id', params, ctx)

print('Found %d Project(s) with ID %d in group private-1' %
      (len(projects), private_project_id))

# Query for the private project we created using read-only-1
#
# You will notice that this does not return the Project as we have **NOT**
# specified the group that the Project is in within the context
# passed to the query service.
ctx = {'omero.group': str(read_only_group.id.val)}
params = ParametersI()
params.addId(private_project_id)
projects = query_service.findAllByQuery(
    'select p from Project as p ' \
    'where p.id = :id', params, ctx)

print('Found %d Project(s) with ID %d in group read-only-1' %
      (len(projects), private_project_id))

# Use the OMERO 4.3.x introduced delete service to clean up the Projects
# we have just created.
handle = session.submit>Delete('/Project', private_project_id, None)
try:
    callback = CmdCallbackI(client, handle)
    callback.loop(10, 1000) # Loop a maximum of ten times each 1000ms
finally:
    # Safely ensure that the Handle to the delete request is cleaned up,
    # otherwise there is the possibility of resource leaks server side that
    # will only be cleaned up periodically.

```

(continues on next page)

(continued from previous page)

```

        handle.close()
handle = session.submit(Delete('/Project', read_only_project_id, None))
try:
    callback = CmdCallbackI(client, handle)
    callback.loop(10, 1000) # Loop a maximum of ten times each 1000ms
finally:
    handle.close()

```

Example output:

```

Created Project:113 in group private-1
Created Project:114 in group read-only-1
Found 1 Project(s) with ID 113 in group private-1
Found 0 Project(s) with ID 113 in group read-only-1

```

Advanced queries

In OMERO 4.4.0, cross group querying was reintroduced. Again, we make use of the Ice context object. Through this object, and manipulations thereof, we can expand our query context to span all groups via the use of `-1`. What follows is a series of example queries using contexts that span all groups.

Querying data across groups

```

#!/python
from omero.model import *
from omero.rtypes import *
from omero.sys import ParametersI
from omero.cmd import Delete, DoAll
from omero.callbacks import CmdCallbackI

# Session that has already been created for user-2
session = client.getSession()

# Project object instantiation
private_project = ProjectI()
private_project.name = rstring('private-1 project')
read_only_project = ProjectI()
read_only_project.name = rstring('read-only-1 project')

# Retrieve the services we are going to use
update_service = session.getUpdateService()
admin_service = session.getAdminService()
query_service = session.getQueryService()

# Groups we are going to write data into
private_group = admin_service.lookupGroup('private-1')
read_only_group = admin_service.lookupGroup('read-only-1')

# Save and return our two projects, setting the context correctly for each.

```

(continues on next page)

(continued from previous page)

```

# ALL interactions with the update service where NEW objects are concerned
# must be passed an explicit context and NOT '-1'. Otherwise the server
# has no idea which set of owners to assign to the object when persisted.
ctx = {'omero.group': str(private_group.id.val)}
private_project = update_service.saveAndReturnObject(private_project, ctx)
ctx = {'omero.group': str(read_only_group.id.val)}
read_only_project = update_service.saveAndReturnObject(read_only_project, ctx)

private_project_id = private_project.id.val
read_only_project_id = read_only_project.id.val
print('Created Project:%d in group private-1' % (private_project_id))
print('Created Project:%d in group read-only-1' % (read_only_project_id))

# Query for the private project we created using private-1
#
# You will notice that this returns both Projects as we have specified
# '-1' in the context passed to the query service.
ctx = {'omero.group': '-1'}
params = ParametersI()
params.addIds([private_project_id, read_only_project_id])
projects = query_service.findAllByQuery(
    'select p from Project as p ' \
    'where p.id in (:ids)', params, ctx)

print('Found %d Project(s)' % (len(projects)))

# Use the OMERO 4.3.x introduced delete service to clean up the Projects
# we have just created. The delete service uses '-1' by default for all its
# internal queries. We are also introducing the 'DoAll' command, which
# allows for the aggregation of 'Delete' commands.
delete_requests = [
    Delete('/Project', private_project_id, None),
    Delete('/Project', read_only_project_id, None)
]
handle = session.submit(DoAll(delete_requests))
try:
    callback = CmdCallbackI(client, handle)
    callback.loop(10, 1000) # Loop a maximum of ten times each 1000ms
finally:
    # Safely ensure that the Handle to the delete request is cleaned up,
    # otherwise there is the possibility of resource leaks server side that
    # will only be cleaned up periodically.
    handle.close()

```

Example output:

```

Created Project:117 in group private-1
Created Project:118 in group read-only-1
Found 2 Project(s)

```


Querying data across users in the same group

Through the use of an `omero.sys.ParametersI` filter, restricting a query to a given user is possible. For the purposes of these examples, we will assume that both `user-2` and `user-3` have a single project each in the `read-only-1` group.

```
#!/python
from omero.model import *
from omero.rtypes import *
from omero.sys import ParametersI

# Session that has already been created for user-2
session = client.getSession()

# Retrieve the services we are going to use
admin_service = session.getAdminService()
query_service = session.getQueryService()

# Groups we are going to query
read_only_group = admin_service.lookupGroup('read-only-1')

# Users we are going to query
user_2 = admin_service.lookupExperimenter('user-2')
user_3 = admin_service.lookupExperimenter('user-3')

# Print the members of 'read-only-1'
print('Members of "read-only-1" (experimenter_id, username): %r' %
      [(v.id.val, v.omeName.val) for v in read_only_group.linkedExperimenterList()])

# Query for all projects
ctx = {'omero.group': str(read_only_group.id.val)}
projects = query_service.findAllByQuery(
    'select p from Project as p', None, ctx)
print('All projects in "read-only-1" (project_id, owner_id): %r' %
      [(v.id.val, v.details.owner.id.val) for v in projects])

# Query for projects owned by 'user-2'
ctx = {'omero.group': str(read_only_group.id.val)}
params = ParametersI()
params.addId(user_2.id.val)
projects = query_service.findAllByQuery(
    'select p from Project as p ' \
    'where p.details.owner.id = :id', params, ctx)
print('Projects owned by "user-2" in "read-only-1" (project_id, owner_id): %r' %
      [(v.id.val, v.details.owner.id.val) for v in projects])

# Query for projects owned by 'user-3'
ctx = {'omero.group': str(read_only_group.id.val)}
params = ParametersI()
params.addId(user_3.id.val)
projects = query_service.findAllByQuery(
    'select p from Project as p ' \
    'where p.details.owner.id = :id', params, ctx)
print('Projects owned by "user-3" in "read-only-1" (project_id, owner_id): %r' %
```

(continues on next page)

(continued from previous page)

```
[(v.id.val, v.details.owner.id.val) for v in projects])
```

Example output:

```
Members of "read-only-1" (experimenter_id, username): [(10L, 'user-2'), (9L, 'user-3')]
All projects in "read-only-1" (project_id, owner_id): [(4L, 10L), (7L, 9L)]
Projects owned by "user-2" in "read-only-1" (project_id, owner_id): [(4L, 10L)]
Projects owned by "user-3" in "read-only-1" (project_id, owner_id): [(7L, 9L)]
```

Utilizing the Permissions object

Every object that is retrieved from the server via the query service, regardless of the context used, has a fully functional `omero.model.PermissionsI` object. This object contains various methods to allow the caller to interrogate the operations that are possible by the current user on the object:

- `canAnnotate()`
- `canChgrp()`
- `canChown()`
- `canDelete()`
- `canEdit()`
- `canLink()`

Troubleshooting permissions issues

Data disappears after a change of the primary group of a user

As outlined above, changes were made so that by default queries do not span multiple groups and the 'Primary or Default Group' is essentially a deprecated concept. If you have multiple groups and you are attempting to make queries by switching the 'Active Group' via the `setSecurityContext()` method of an active session (`omero.cmd.SessionPrx`), those queries will be scoped only to that group. If you want your queries to act more like they did in 4.1.x, setting `omero.group=-1` will achieve that.

However, the reasons we made these changes have more to them than just API usage and the OMERO client history of only showing the data from one group at a time. Changing the 'Active Group' is both expensive because of the atomicity requirements the server enforces and can create dangerous concurrency situations. This is further complicated by the addition of the change group and delete background processes since 4.1.x. Manipulating a session's 'Primary or Default Group' during these tasks can have drastic effects. Changing the 'Active Group' is forbidden if there are any stateful services (`omero.api.RenderingPrx` for example) currently open.

In short, in OMERO 5.6.10 you absolutely **should not** be switching the 'Primary or Default Group' of the user, or the 'Active Group' of a session, as a means to achieve cross group querying.

Listing other users' data in read-only groups

In order to list other users' data associated with read-only groups of which you are a member, you can also use the context object and set the `omero.group` to `-1`. In addition, you can add a filter to the query to only select the other users' data. You can do this either by using the `omero.sys.ParametersI` object's `exp()` method when using the `IContainer` service, or by an explicit query when using `IQuery` service.

Is the default group the primary group when not specifying the context?

The value of the `groupId` property of the `omero.sys.EventContext` is the "Active Group" for the created session. It can be modified as described above with the restrictions outlined. Unless the session has been created by means other than `createSession()` on an `omero.client` object, this will be the user's "Primary or Default Group." A user's 'Primary or Default Group' is the first group in the collection that describes the relation `Experimenter <--> ExperimenterGroup`. It can be set by the `setDefaultGroup()` method on the `IAdmin` service.

What about when importing data without specifying the context object?

Exactly as outlined above. Import does nothing different or special. If you want the operating context of an import to be different from the default you must specify it as such.

Specifying the group context as -1 when deleting data

There is no need to do this. Complete graphs cannot span multiple groups and queries are only (unless otherwise filtered) restricted at the group level and not at the level of the user. Furthermore, the delete service always internally performs all its queries in the `omero.group=-1` context unless another more explicit one is specified.

History

The OMERO permissions model has had a significant overhaul from version 4.1.x to 4.4.x. Users and groups have existed in OMERO since well before the initial 4.1.x releases and numerous permissions levels were possible in the 4.1.x series but it was largely assumed that an `Experimenter` belonged to a single `Group` and that the permissions of that `Group` were private.

The OMERO permissions system received its first significant update in 4.2.0 with the introduction of multiple group support throughout the platform and group permissions levels.

In a 4.1.x object graph `Group` containment was not enforced i.e. two linked objects (such as a `Project` and `Dataset`) could in theory be members of two distinct `Groups`. All objects continued to carry their permissions and those permissions were persisted in the database.

Things to note about 4.2.x permissions

- Objects could not be moved between groups easily.
- It was not possible to reduce the permissions level of a group.
- The delete service (introduced in OMERO 4.2.1) was made aware of the permissions system.
- 'Default Group' switching was required to make queries in different permissions contexts.

Note: Queries span only one group at a time. Inserts and updates as other users must be done by creating a session as that user.

Changes for OMERO 4.4.x

The second major OMERO permissions system innovations were performed in 4.4.0:

- Cross group querying was reintroduced.
- Change group was enabled, allowing the movement of graphs of objects between groups.
- Permissions level reduction was made possible for read-annotate to read-only transitions.
- **A thorough user interface review resulted in the following features being made available in the UI:**
 - single group browsing and user-switching (available since 4.4.0)
 - browsing data across multiple groups (available since 4.4.6 and refined in 4.4.7)
- The concept of a ‘Default or Primary Group’ was deprecated.

Note: Queries, inserts and updates span any or all groups and any user via options flags.

Changes for OMERO 5.4.x

OMERO 5.4.0 included Restricted Administrators as a new user role. See *Administrators with restricted privileges* and *The server’s view of administrator restrictions* for more information.

3.11 OMERO.server in depth

3.11.1 OMERO.server overview

OMERO sequence narrative

Trying to understand all of what goes on with the server can be a bit complicated. This short narrative tries to touch on the most critical aspects.

- A request reaches the server over one of the two remoting protocols: RMI or ICE. First, the [Principal](#) is examined for a valid [session](#) which was created via `ISession.createSession(String username, String password)`.
- These values are checked against the `experimenter`, `experimentergroup` and `password` tables. A valid login consists of a user name which is to be found in the `omename` column of `experimenter`. This row from `experimenter` must also be contained in the “user” experimenter group which is done via the mapping table `groupexperimentermap` (see [this SQL template](#) for how root and the initial groups are setup).
- If the server is configured for [LDAP Authentication](#), an `Experimenter` may be created when `ISessions` attempts to check the password via `IAdmin.checkPassword()`.
- If authentication occurs, the request is passed to an [EJB3](#) interceptor which checks whether or not the authenticated user is authorized for that service method. Methods are labelled either `@RolesAllowed("user")`, `@RolesAllowed("system")`, or `@PermitAll`. All users are a member of “user”, but only administrators will be able to run “system” methods.

- If authorization occurs, the request finally reaches a container-managed stateful or stateless *service*. The service will *prepare* the OMERO runtime for the particular user – checking method parameters, creating a new *event*, initializing the *security system*, etc. – and pass execution onto the method implementation. This is done using references acquired (or injected) from the Spring *application context*.
- The actual service implementation (from `ome.logic` or `ome.services`) will be either read-only (*IQuery*-based) or a read-write (*IUpdate*-based).
- In the case of a read-only action, the implementation asks the database layer for the needed object graph, transforms them where necessary, and returns the values to the remoting subsystem. On the client-side, the returned graph can be mapped to an internal model via the ((OMERO Model Mapping|model wrapper)).
- In the case of a read-write action, the change to the database is first passed to a validation layer for extensive checking. Then the graph is passed to the database layer which prepares the SQL, including an audit trail of the changes made to the database.
- After execution, the OMERO runtime is reset, the method call is logged, and either the successful results are returned or an *exception* is thrown.

Technologies

It is fairly easy to work with the server without understanding all of its layers. The API is clearly outlined in the `ome.api` package and the client proxies work almost as if the calls were being made from within the same virtual machine. The only current caveat is that objects returned between two different calls will not be referentially (i.e. `obj1 == obj2`) equivalent. We are working on removing this restriction.

To understand the full technology stack, however, there are several concepts which are of importance:

- A layered architecture ensures that components only “talk to” the minimum necessary number of other components. This reduces the complexity of the entire system. Ensuring a loose-coupling of various components is facilitated by dependency injection. Dependency injection is the process of allowing a managing component to place a needed resource in a component’s hand. Code for lookup or creation of resources, in turn, is unneeded, and explicit implementation details do not need to be hard-coded.
- Object-relational mapping (ORM) is the process of mapping relational tables to object-oriented classes. Currently OMERO uses *Hibernate* to provide this functionality. ORM allows the developer to work in a known environment, here the type-safe world of Java, rather than writing difficult to debug sql.
- Aspect-oriented programming, a somewhat new and misunderstood technology, is perhaps the last technology which should be mentioned. Various pieces of code (“aspects”) are *inserted* at various moments (“joinpoints”) of execution. Collecting logic into aspects, whether logging, transactions, security etc., also reduces the overall complexity of the code.

Server design

The server logic resides in the `server` component.

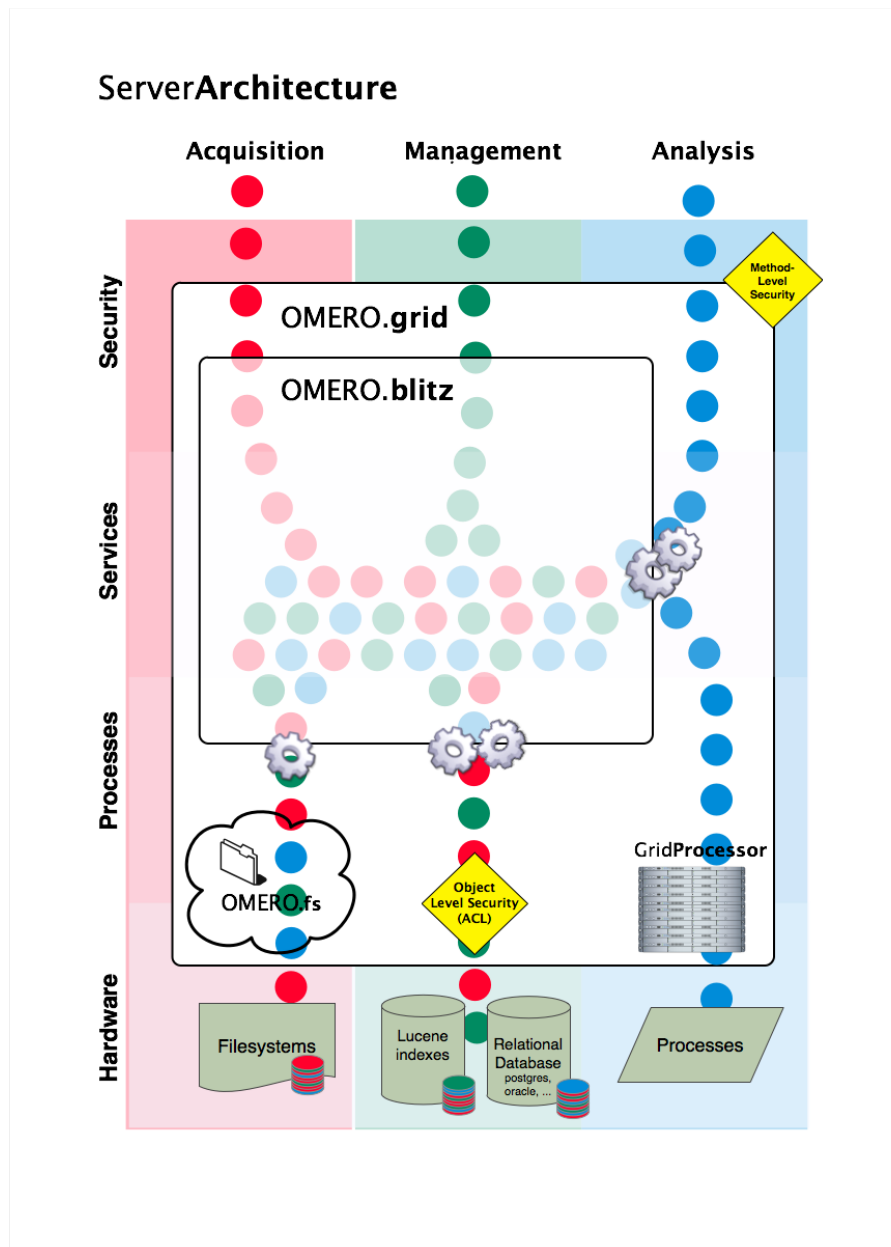


Fig. 12: Server Architecture

ServerDesign

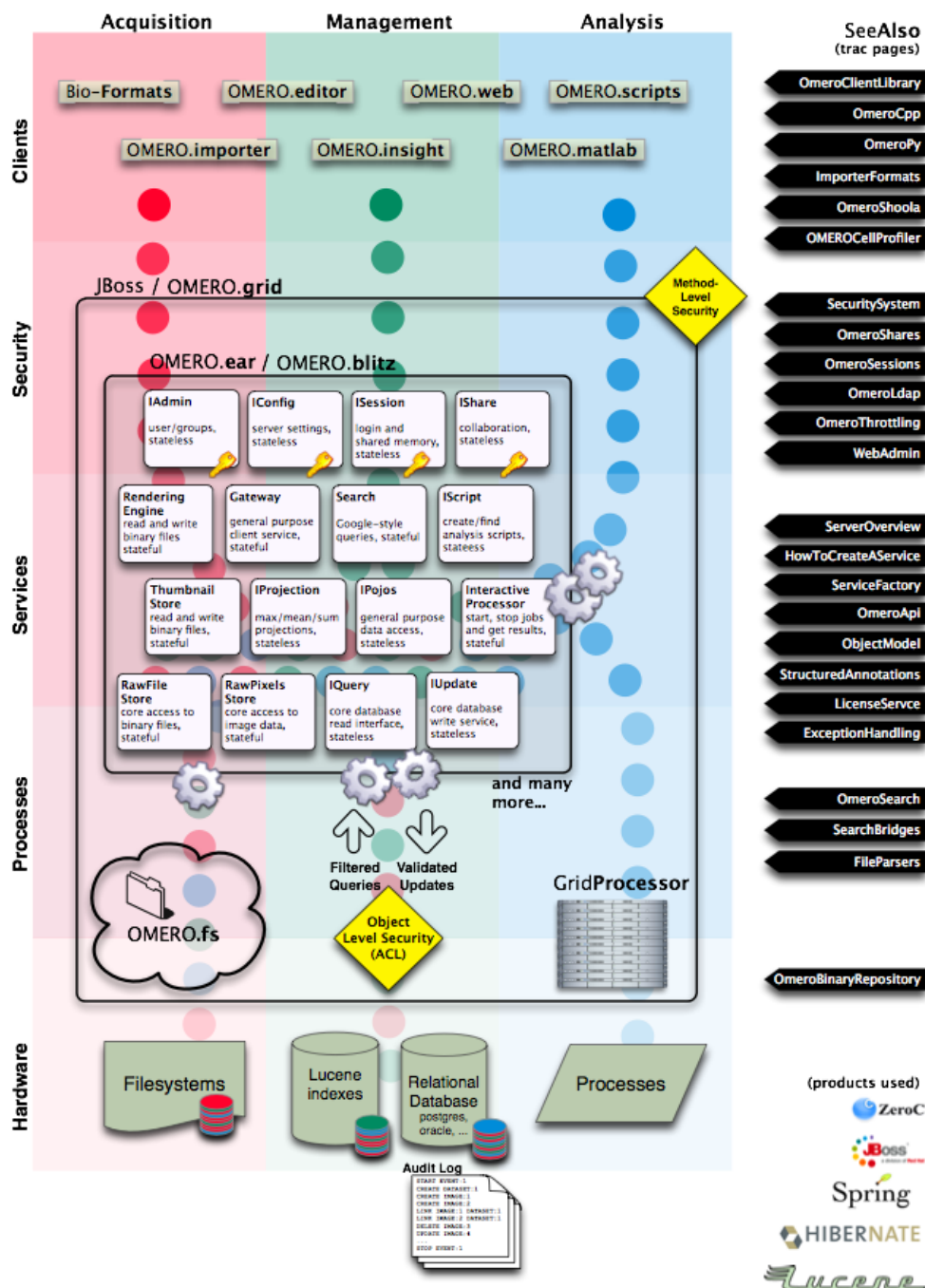


Fig. 13: Server Design

Topics

- *Exception handling*
- *OME-Remote Objects*
- *Server security and firewalls*

See also:

OMERO.grid

3.11.2 Extending OMERO.server

Overview

Despite all the effort put into building OMERO, it will never satisfy the requirements of every group. Where we have seen it useful to do so, we have created extension points which can be used by third-party developers to extend, improve, and adapt OMERO. We outline most of these options below as well as some of their trade-offs. We are also always interested to hear other possible extension points. Please contact the [forum](#) with any such suggestions.

Existing extension points

To get a feeling for what type of extension points are available, you might want to take a look at the following pages. Many of them will point you back to this page for packaging and deploying your new code.

- *File parsers* - write Java file parsers to further extend search
- *LoginAttemptListener* - write a Java handler for failed login attempts
- *Command Line Interface as an OMERO development tool* - write drop in Python extensions for the command-line
- *Introduction to OMERO.scripts* - write python scripts to process data server-side
- *LDAP plugin design* - write a Java authentication plugin
- *Password Provider* - write a Java password backend
- *Search bridges* - write Java Lucene parsers to extend search
- *OMERO.mail* - server email sender (added in OMERO 5.1, no developer documentation as yet)
- *omero.policy.bean* - policy configuration point e.g. for setting download restriction policy on users (added in OMERO 5.1, no developer documentation as yet)

Extending the Model

The OME Data Model and its OMERO representation, the *OME-Remote Objects*, intentionally draw lines between what metadata can be supported and what cannot. Though we are always examining new fields for inclusion, it is not possible to represent everyone's model within OME.

Structured annotations

The primary extension point for including external data are the *Working with annotations* (SAs). SAs are designed as email-like attachments which can be associated with various core metadata types. In general, they should link to information outside of the OME model, i.e. information which OMERO clients and servers do not understand. URLs can point to external data sources, or XML in a non-OME namespace can be attached.

The primary drawbacks are that the attachments are opaque and cannot be used in a fine-grain manner.

Code generation

Since it is prohibitive to model full objects with the SAs, one alternative is to add types directly to the *generated code*. By adding a file named `*.ome.xml` to <https://github.com/ome/omero-model/tree/v5.6.13/src/main/resources/mappings> and running a full-build, it is possible to have new objects generated in all *OMERO.blitz* languages. Supported fields include:

- boolean
- string
- long
- double
- timestamp
- links to any other `ome.model.*` object, including enumerations

For example:

```
<types>
  <!-- "named" and "described" are short-cuts to generate the fields "name" and
  ↳ "description" -->
  <type id="ome.model.myextensions.Example" named="true" described="true">
    <required name="valueA" type="boolean"/> <!-- This is NONNULL -->
    <optional name="valueB" type="long"/> <!-- This is nullable -->
    <onemany name="images" type="ome.model.core.Image"/> <!-- A set of images -->
  </type>
</types>
```

Collections of primitive values like `<onemany name="values" type="long"/>` are not supported. Please see the existing mapping files for more examples of what can be done.

The primary drawback of code-generating your own types is isolation and maintenance. Firstly, your installation becomes isolated from the rest of the OME ecosystem. New types are not understood by other servers and clients, and cannot be exported or shared. Secondly, you will need to maintain your own server **and** client builds of the system, since the provided binary builds would not have your new types.

Measurement results

For storing large quantities of only partially structured data, such as tabular/CSV data with no pre-defined columns, neither the SAs nor the code-generation extensions are ideal. SAs cannot easily be aggregated, and code-generation would generate too many types. This is particularly clear in the storage and management of HCS analysis results.

To solve this problem, we provide the *OMERO.tables* API for storing tabular data indexed via Roi, Well, or Image id.

Services

Traditionally, services were added via Java interfaces in the <https://github.com/ome/omero-common/tree/v5.6.4/src/main/java/ome/api> package. The creation of such “core” services is described under *How To create a service*. However, with the introduction of *OMERO.blitz*, it is also possible to write blitz-only services which are defined by a slice definition under <https://github.com/ome/omero-blitz/tree/v5.7.2/src/main/slice/omero>.

A core service is required when server internal code should also make use of the interface. Since this is very rarely the case for third-party developers wanting to extend OMERO, only the creation of blitz services will be discussed here.

Add a slice definition

The easiest possible service definition in slice is:

```
module example {  
    interface NewService {  
        void doSomething();  
    };  
};
```

This should be added to any existing or a new *.ice file under the `src/main/slice/omero` directory. After the next ant build, stubs will be created for all the *OMERO.blitz* languages, i.e. *OMERO Java language bindings*, *OMERO Python language bindings*, and *OMERO C++ language bindings*.

Note: Once you have gotten your code working, it is most re-usable if you can put it all in a single directory under `tools/`. These components also have their `resources/*.ice` files turned into code, and they can produce their own artifacts which you can distribute without modifying the main code base.

Warning: exceptions

You will need to think carefully about what exceptions to handle. Ice (especially *OMERO C++ language bindings*) does not handle exceptions well that are not strictly defined. In general, if you would like to add your own exception type, feel free to do so, but either 1) subclass `omero::ServerError` or 2) add to the appropriate `throws` clauses. And regardless, if you are accessing any internal OMERO API, add `omero::ServerError` to your `throws` clause.

See *Exception handling* for more information.

Java implementation using `_Disp`

To implement your service, create a class subclassing “example._NewServiceDisp” class which was code-generated. In this example, the class would be named “NewServiceI” by convention. If this service needs to make use of any of the internal API, it should do so via dependency injection. For example, to use `IQuery` add either:

```
void setLocalQuery(LocalQuery query) {
    this.query = query;
}
```

or

```
NewServiceI(LocalQuery query) {
    this.query = query;
}
```

The next step “Java Configuration” will take care of how those objects get injected.

Java implementation using `_Tie`

Rather than subclassing the `_Disp` object, it is also possible to implement the `_Tie` interface for your new service. This allows wrapping and testing your implementation more easily at the cost of a little indirection. You can see how such an object is configured in [blitz-servantDefinitions](#).

Java configuration

Configuration in the Java servers takes place via [Spring](#). One or more files matching a pattern like `ome/services/blitz-*.xml` should be added to your application.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/
↪spring-beans.dtd">
<beans>

    <bean class="NewServiceI">
        <description>
            This is a simple bean definition in Spring. The description is not necessary.
        </description>
        <constructor-arg ref="internal-ome.api.IQuery"/>
    </bean>

</beans>
```

The three patterns which are available are:

- `ome/services/blitz-*.xml` - highest-level objects which have access to all the other defined objects.
- `ome/services/services-*.xml` - internal server objects which do not have access to `blitz-*.xml` objects.
- `ome/services/db-*.xml` - base connection and security objects. These will be included in background java process like the index and pixeldata handlers.

Note: *Password Provider* and similar should be included at this level.

See <https://github.com/ome/omero-blitz/tree/v5.7.2/src/main/resources/ome/services> and <https://github.com/ome/omero-server/tree/v5.6.10/src/main/resources/ome/services> for all the available objects.

Java deployment

Finally, these resources should all be added to `OMERO_DIST/lib/server/extensions.jar`:

- the code generated classes
- your `NewServiceI.class` file and any related classes
- your `ome/service/blitz-*.xml` file (or other XML)

Non-service beans

In addition to writing your own services, the instructions above can be used to package any Spring-bean into the OMERO server. For example:

```
//
// MyLoginAttemptListener.java
//
import ome.services.messages.LoginAttemptMessage;

import org.springframework.context.ApplicationListener;

/**
 * Trivial listener for login attempts.
 */

public class MyLoginAttemptListener implements
    ApplicationListener<LoginAttemptMessage> {

    public void onApplicationEvent(LoginAttemptMessage lam) {
        if (lam.success != null && !lam.success) {
            // Do something
        }
    }

}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/
→spring-beans.dtd">
<!--
//
// ome/services/blitz-myLoginListener.xml
//
-->
<beans>
```

(continues on next page)

(continued from previous page)

```
<bean class="myLoginAttemptListener" class="MyLoginAttemptListener">
  <description>
    This listener will be added to the Spring runtime and listen for all
    ↪LoginAttemptMessages.
  </description>
</bean>

</beans>
```

Putting `MyLoginAttemptListener.class` and `ome/services/blitz-myLoginListener.xml` into `lib/server/extensions.jar` is enough to activate your code:

```
~/example $ ls -l
MyLoginListener.class
MyLoginListener.java
lib
...
~/example $ jar cvf lib/server/extensions.jar MyLoginListener.class ome/services/blitz-
↪myLoginListener.xml
added manifest
adding: MyLoginListener.class(in = 0) (out= 0)(stored 0%)
adding: ome/services/blitz-myLoginListener.xml(in = 0) (out= 0)(stored 0%)
```

Servers

With the *OMERO.grid* infrastructure, it is possible to have your own processes managed by the OMERO infrastructure. For example, at some sites, *NGINX* is started to host *OMERO.web framework*. Better integration is possible however, if your server also uses the *Ice* remoting framework.

One way or the other, to have your server started, monitored, and eventually shutdown by *OMERO.grid*, you will need to add it to the “application descriptor” for your site. When using:

```
omero admin start
```

the application descriptor used is `etc/grid/default.xml`. The `<application>` element contains various `<node>`s. Each node is a single daemon process that can start and stop other processes. Inside the nodes, you can either directly add a `<server>` element, or in order to reuse your description, you can use a `<server-instance>` which must refer to a `<server-template>`.

To clarify with an example, if you have a simple application which should watch for newly created Images and send you an email: `mail_on_import.py`, you could add this in either of the following ways:

Server element

```
<node name="my-emailer-node"> <!-- this could also be an existing node, but it must be
↪unique -->
  <server id="my-emailer-server" exe="/home/josh/mail_on_import.py" activation="always">
    <env>${PYTHONPATH}</env>
    <!-- The adapter name must also be unique -->
    <adapter name="MyAdapter" register-process="true" endpoints="tcp"/>
```

(continues on next page)

(continued from previous page)

```
</server>
</node>
```

Server-template and server-instance elements

```
<server-template id="emailer-template"> <!-- must also be unique -->
  <property name="user"/>
  <server id="emailer-server-#{user}" exe="/home/#{user}/mail_on_import.py" activation=
  ↪ "always">
    <env>${PYTHONPATH}</env>
    <adapter name="MyAdapter" register-process="true" endpoints="tcp"/>
  </server>
</server-template>

<node name="our-emailer-node">
  <server-instance id="emailer-template" user="ann">
  <server-instance id="emailer-template" user="ann">
</node>
```

See also:

[ome-devel] model description driven code generation

3.11.3 OMERO.blitz

The OMERO.blitz server is responsible for providing secure access to data and metadata via user sessions (*OMERO sessions*), and cleaning up all resources when they are no longer being used. Various server capabilities are accessed via a multitude of services collectively known as the *OMERO Application Programming Interface*.

Metadata

Metadata stored in an object-relational database is mapped into the OMERO *OME-Remote Objects* via *Hibernate*. *Hibernate Query Language (HQL)* calls can be made against the server and have all ownership information automatically taken into account.

Image data

The binary image data can either be accessed in its raw form via the *RawPixelsStore* service, or can be rendered by the *OMERO.server image rendering* service.

3.11.4 OMERO.fs

OMERO.fs is a series of on-going changes designed to improve the way an OMERO.server interacts with existing directories of acquired image data. It currently consists of two components:

OMERO.dropbox is designed for watching a directory and kicking off an automatic import. The configuration of the DropBox system is covered on the [OMERO.dropbox](#) system administrator's page.

OMERO.fs Managed Repository is designed to store original data in an unaltered form without requiring the data duplication that was carried out by a pre-5.0 import. The changes to the import system mean that an OMERO.fs server stores the original files in the **ManagedRepository**, preserving file names and any nested directory structure. The repository may even contain direct *in-place links* to original data without an file upload step. These changes improve the way OMERO deals with High Content Screening (HCS) and other complex heterogeneous data types, reducing storage requirements and using Bio-Formats to recognize *Filesets* (groups of files which correspond to multi-dimensional images and accompanying information) so they can be treated as single entities within the OMERO clients. OMERO.fs has some *configuration properties* that allow sysadmins to customize it for their site; developers should be aware of the *how the new import process works* and how this is affected by the configuration.

3.11.5 Import under OMERO.fs

The OMERO 5 release introduces OMERO.fs, a new way of storing files in the OMERO binary repository and thus a new method of importing images to the server.

In previous versions of OMERO the import process was very much client-centered. When importing an image the client pushed pixel data, metadata and, optionally, original image files to the OMERO server. With the advent of OMERO 5, OMERO.fs allows the pixels to be accessed directly from the original image files. This means that much of the import process can now take place on the server once the original image files have been uploaded.

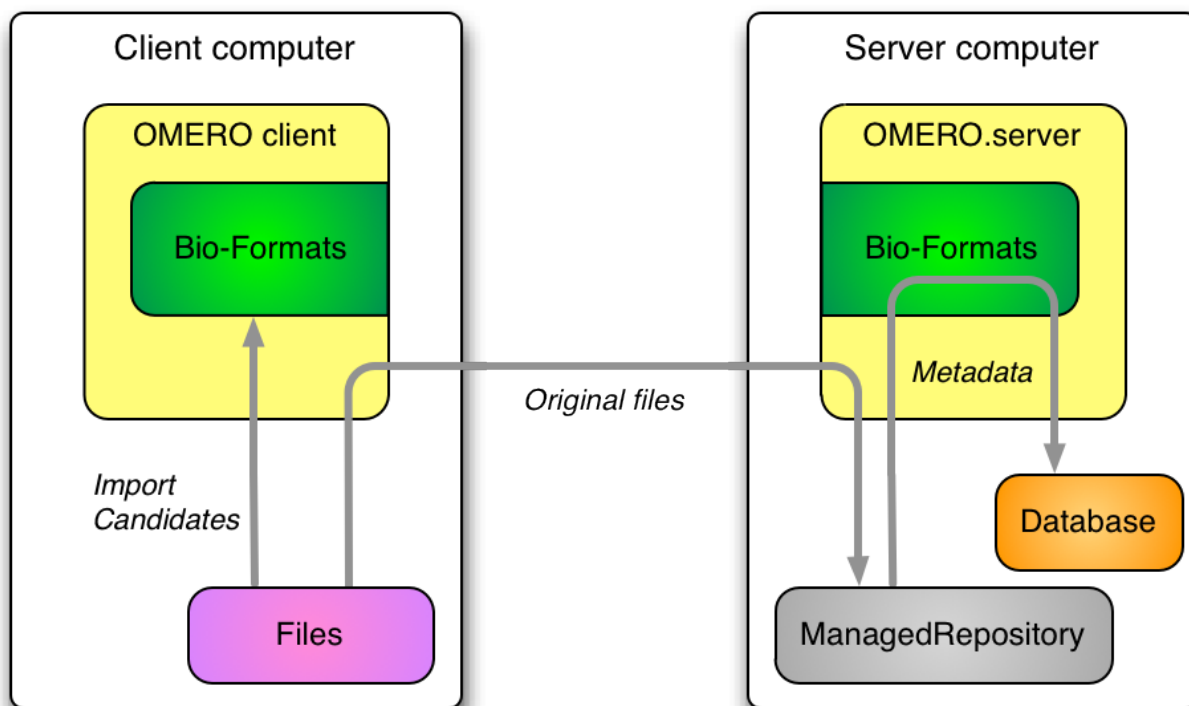
This page looks at the implications for the developer writing import clients. A broad description of the import workflow is followed by some of the model changes needed to facilitate this workflow. The current API sequence is then introduced before looking at server-side classes and sequence. Finally, the configuration required for OMERO.fs is specified.

Import overview

The broad import workflow comprises selecting a file or set of files to be imported client-side. Using Bio-Formats on the client this selection is resolved into a number of import candidates. Here an import candidate is a file or set of files that represents a single image, a multi-image set or a plate. Each import candidate, which may be one file or several files, is then treated as **Fileset** for import. The import of each **Fileset** is then undertaken by the client in two stages: upload and server-side import.

A **Fileset** is uploaded to the server into a location determined by the server, multiple **Filesets** may be uploaded in parallel by a client. A checksum is calculated before upload by the client and after upload by the server. If these checksums match then the client triggers a server-side import. The client can then move on to doing other work and leave the import to complete on the server.

Once the **Fileset** is on the server and an import has been initiated by the client all processing then takes place on the server. The server then uses Bio-Formats to extract and store the metadata, calculate the minimum and maximum pixel values and do other import processing.



Filesets

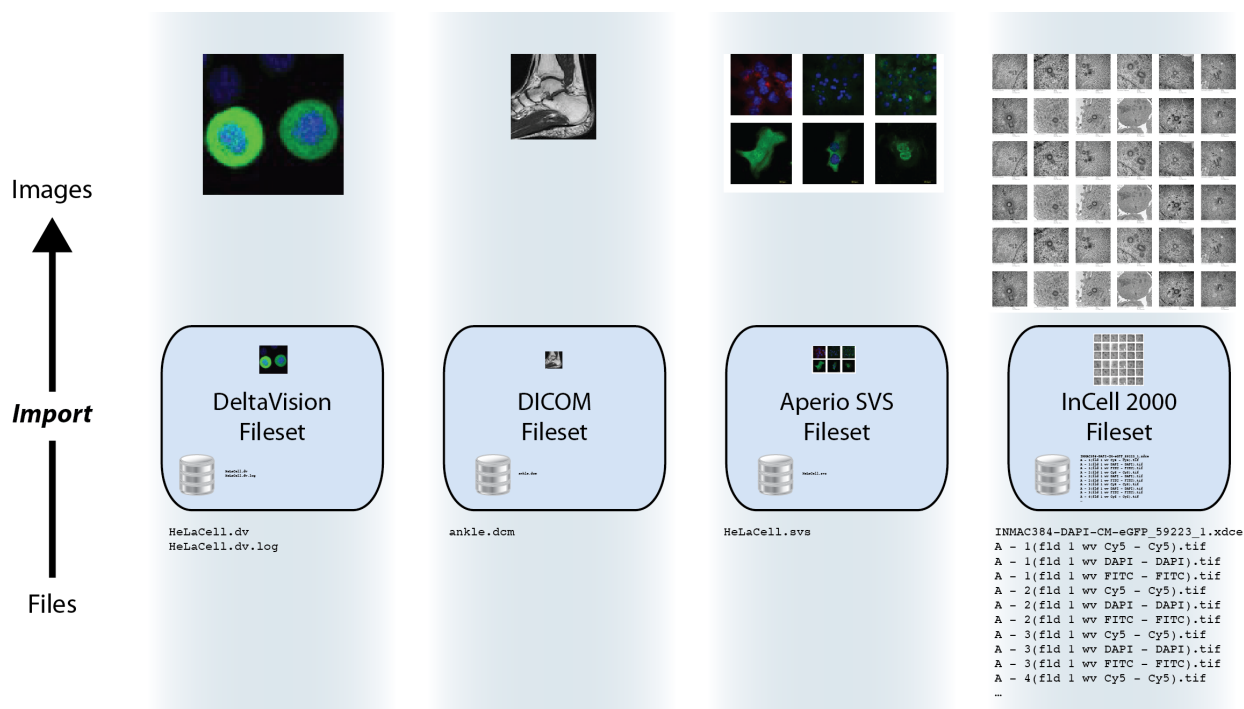
A **fileset** is a concept new to Omero.fs which captures how **Bio-Formats** relates files to the images that they encode. If importing a single file leads to a single corresponding image being viewable then a one-to-one mapping exists from file to image. However, an image's information may be split among multiple files, or a single file may encode multiple images. In other cases, especially in high-content screening, many images of wells may be encoded in a complex manner by many files. In all these cases, a fileset is used to hold the set of files and the set of images to which those files correspond.

Filesets are essentially indivisible: the files or images that a fileset represents are deleted, or moved between groups, only as one unit together, and on the server each fileset has a directory in which only its files are stored. Each fileset firmly binds sets of files and images because the dependencies among them mean that **splitting components away leaves a partial fileset making the remaining data unreadable**. Similarly, **you should not rename any components of a fileset** as this will also break the dependencies holding them together.

Model description

See [acquisition.ome.xml](#).

- `ome.model.fs.Fileset`
 - Represents a group of files which are considered to belong together.
 - In the future the client may upload a single “import set” that is subsequently resolved by Bio-Formats into multiple filesets on the server.
 - Also links to multiple `ome.model.jobs.Job` instances.
 - Links to the image objects that are created during import.
- `ome.model.jobs.Job` subclasses



- Each one represents some action which takes place server-side on the Fileset.
- For the standard sequence described above, the first will always be an `UploadJob` which contains version info from the client. If the files were not uploaded however, this may not be the case. Then a `MetadataImportJob` follows, which is the basic import.
- Other jobs may be necessary for regular usage (`PixelDataJob`, `IndexingJob`, etc.). *Later jobs may also be added like a re-parse job, a re-check of the hashes to detect corruption, or an archive job.*
- For job definitions, see [jobs.ome.xml](#).
- Some subclasses have a `versionInfo` property for storing a snapshot of process information along with software versions. Most important for knowing how files were parsed, therefore when using `importPaths`, a “synthetic” version info might be created to say that these were just uploaded blindly.
- `ome.model.fs.FilesetEntry`
 - Link from a `Fileset` to exactly one `ome.model.core.OriginalFile`
 - Critically, it also contains the original absolute client path of that file.

API sequence

- Choose which files to import by either:
 - `ImportLibrary` and friends (Java only)
 - listing all files (not directories) manually.
- Choose a `ManagedRepositoryPrx` from `SharedResourcesPrx.repositories()`.
- Call either:
 - `ImportLibrary.importImage()` which calls `ManagedRepositoryPrx.importFilesets(Fileset, ImportSettings)`, or

- directly use `ManagedRepositoryPrx.importPaths(StringSet)`.
- Receive an `ImportProcessPrx`.
- For each `FileEntry` in the `FileSet` or each path in the `StringSet` (in order), call `ImportProcessPrx.getUploader()` and receive a `RawFileStorePrx`.
- Upload the file via `RawFileStorePrx.write()` while reading the files locally to write, be sure to calculate the checksum.
- Pass a list of checksums (in order) to `ImportProcessPrx.verifyUpload(StringSet)`. If the hashes match, receive a `HandlePrx`. Otherwise an exception is thrown.

At this point, the client should be able to disconnect and the rest of the import should happen independently.

- Create an `CmdCallbackI` that wraps the `HandlePrx` and wait for successful completion.

At this point, the main metadata import is finished, but background processing may still be occurring. Handles for the background processing will also be returned.

Server-side classes/concepts

`AbstractRepositoryI` and all of its subclasses are implementations of the `InternalRepository` API. These objects are for internal use only and should never be accessible by the clients. Each instance is initialized with a directory which the servant attempts to “acquire” (i.e. grab a lock file). Once it does so, it is the serving repository.

Each internal repository provides a public view which in turn provides the `Repository` API. All method calls assume Unix-style strings, which are guaranteed by `CheckedPath`, a loose wrapper around `java.io.File`. `CheckedPath` objects along with the active `Ice.Current` instance are passed to the `RepositoryDao` interface, which provides database access for all repositories. Access to raw IO is provided by the `RepoRawFileStoreI` servant, which wraps a `RawFileBean`.

The `ManagedRepository` implementation is responsible for import and enforces further constraints (beyond those of `CheckedPath`) on where and what files are created. Most importantly, the `omero.fs.repo.path` template value is expanded and pre-pended to all uploads. A further responsibility of the `ManagedRepository` is to maintain a list of all currently running `ManagedImportProcessI`, each of which is held in the `ProcessContainer`. These `ManagedImportProcessI` instances further wrap `RepoRawFileStoreI` instances with a subclass, `ManagedRawFileStoreI`.

For file import through `ManagedRepository.importFileset`, although `hasher` is nullable ordinarily, it will be set through the mandatory `ImportSettings.checksumAlgorithm` property. `ManagedRepository.listChecksumAlgorithms` lists the hashers supported by the server. `ManagedRepository.suggestChecksumAlgorithm` helps the client and server to negotiate a mutually acceptable algorithm, as in `ImportLibrary.createImport`; the result is affected by the server’s configuration setting for `omero.checksum.supported`. `ImportLibrary` calculates each file’s hash using hashers obtained through `ChecksumProviderFactory.getProvider`. In fetching `OriginalFile` objects by HQL through the Query Service one needs `JOIN FETCH` on the hasher property to read the hasher’s name.

Server-side sequence

NB: Server-side `ImportLibrary` is no longer being used. That logic is currently moved to `ManagedImportRequestI`. This may not be the best location. Further, several other layers might also be collapsible, like `OMEROMetadataStore` which is currently accessible as a “hidden” service `MetadataStorePrx`. Here, “hidden” means that it is not directly retrievable from `ServiceFactoryPrx`.

- `ManagedRepositoryI.importPaths()`
 - reuses `ImportContainer.fillData()` to create an `ImportSettings` and a `Fileset` and then calls `importFileset(Fileset, ImportSettings)`

- `ManagedRepositoryI.importFileset()`
 - determines an `ImportLocation` calling `PublicRepositoryI.makeDir()` where necessary.
 - `createImportProcess` creates a `ManagedImportProcessI`, registers it, and returns it.
 - After this, the repository is only responsible for periodically having the ping and eventually the shutdown method called, via `ProcessContainer`.
- `ManagedImportProcessI.getUploader()`
 - creates a new `RepositoryRawFileStoreI` for each file in the paths/fileset.
 - Once `close()` is called on this instance, `closeCalled(int i)` will be called on the import process and the instance will be removed.
 - If `getUploader()` is called again, then a new file store will be created.
- `ManagedImportProcessI.verifyUpload()`
 - If all hashes match, then a `ManagedImportRequestI` instance is created and submitted to `omero.cmd.SessionI.submit_async()` for background processing. The client can wait for the returned `omero.cmd.HandlePrx` to finish by using a `CmdCallbackI`.
 - At this point, the `ImportProcessPrx` can be closed as well as the entire client and the import would still continue. Only if `HandlePrx.cancel()` is called, will the import be aborted.
 - QUESTION: How to handle rollback at this point?
- `ManagedImportRequestI.init` (within transaction)
 - `Registry.getInternalServiceFactory()` grabs a `ServiceFactoryPrx` without the need for an `omero.client` instance.
 - `OMEROWrapper` and a `OMEROMetadataStoreClient` are created with this connection.
 - Some other basic configuration takes place.
- `ManagedImportRequest.step()` (N times, each within same transaction as `init()`)
 - NB: it may later make more sense for this bit to happen in a separate process.
 - At the moment, 5 steps are hard-coded. Each performing roughly the same amount of work. Some of these may later be done in the background.
 - * `importMetadata()` calls `store.saveToDB()`, which calls `MetadataStorePrx.saveToDb()`, a remote call. This could possibly be inlined.
 - * `generateThumbnails()` calls `store.resetDefaultsAndGenerateThumbnails()`, another remote call, which could also be inlined.
 - * `pixelData()` calls `store.setPixelsParams()`, `updatePixels()`, and `populateMinMax()`. Min/Max especially should be backgrounded.
 - * Finally, `store.launchProcessing()` is called, which should remain, but could also be inlined. The returned script processes could be returned in the `ImportResponse`.
 - * Return appropriate values.
 - `notifyObservers()` currently does nothing, since this was client-side functionality in `ImportLibrary`. This needs to be replaced!
- `ManagedImportRequest.buildResponse()` (N times, outside the transaction)
 - Only step 4 does anything, storing the pixels in a `ImportResponse`
- `ManagedImportRequest.getResponse()` (1 time, regardless of exception or not)

- Performs cleanup, then returns the `ImportResponse` assuming that no call to `helper.cancel()` has been made. At this point, `ImportLibrary.importImage()` returns successfully.

See also:

FS configuration options

In-place import

3.11.6 OMERO.processor

The Processor is a python process-launcher which can be run on any Unix system to execute scripts for a user. This makes use of the *scripting service* functionality. As many processor nodes can be started as physical computers are available.

- Source code: [processor.py](#)
- Documentation: [OMERO.grid](#)

3.11.7 OMERO.server image rendering

A major requirement for any image data application is the ability to display images. In most applications, this is achieved by reading pixel data from a filesystem and then mapping the pixel data to the 256 grey level available on most computer display monitors. It is common in some experiments to record and display multiple channels at once. Typically three, four, or even five separate images must be mapped, and then presented as a color image for painting on a monitor. Because these operations can require many thousands of operations and must be displayed rapidly to support the display of time-lapse movies, most image display software applications use a high-speed graphics CPU and dedicated hardware for image rendering and display. This requirement limits the deployment of these applications to high-powered workstations.

OMERO.server includes an image server, a software application that delivers rendered images to a client. This ensures that client applications can display image data. The OMERO Rendering Engine (OMERO-RE) has been designed to minimize the amount of data transferred to the client and thus removes the requirement for a specific graphics CPU, allowing high-performance image viewing on standard laptop computers. The OMERO-RE achieves this by limiting data transfer times by being close to the data, using highly efficient network transfer protocols, utilizing modern multi-processor and multi-core machines to provide the data to clients in a format that is as efficient to display as possible. OMERO-RE is multi-threaded and can use multi-core servers to simultaneously render individual channels before assembly into a final color image ready for transfer to the client. The use of the RE is not mandatory. If a client needs to have the full pixel data, it can. This OriginalPixels facility is used for client-side analysis, like that performed in the OMERO.insight measurement tool.

Transfer of image data even after rendering can limit performance, especially when accessing data remotely on connections with limited bandwidth (e.g. domestic ADSL). Therefore the OMERO-RE contains a compression service with an API that allows a client adjustable compression providing minimal image artefacts and a 20-fold range of data size to the client.

The OMERO Rendering Engine is accessed by OMERO client applications written in Java, C++, or Python via a binary protocol (ICE) provided by [ZeroC](#).

3.11.8 Clustering

Clustering an OMERO instance consists of starting multiple *OMERO.blitz* servers with each allocating user sessions based on some criteria. There are at least two reasons you may want to cluster the OMERO server: availability and throughput.

Availability

Having the ability to have two servers up at the same time implies that even if you have to restart one of the servers, there should be no downtime. Currently, *OMERO sessions* are sticky to a cluster node so it is not possible to shut down a node at any time. However, all new sessions can be redirected to the server that is to be left turned on. When all active sessions have completed, the chosen server can be shut down.

Throughput

The other main reason to have other servers running is to service more user sessions simultaneously. When dealing with memory-intensive operations like rendering, each added server can make a positive difference. This is only a part of the story, since much of the bottleneck is not the server itself but other shared resources, like the database or the filesystem. To further extend throughput you will need to parallelize these.

Installation

If you are using the default *OMERO.grid* application descriptor then quickly enabling clustering is as simple as executing:

```
omero config set omero.cluster.redirector configRedirector
omero node backup start
```

This starts a second node, named “backup”, which contains a second *OMERO.blitz* server, “Blitz-1”. By default, this newly created server will not be used until sessions are manually redirected to it.

See also:

Scaling Omero

Read-Only

A read-only server disallows many operations while still permitting users to log in and retrieve data. Prohibitions for users of read-only servers include that they may not import data or run scripts. Two properties control read-only configuration: `omero.cluster.read_only.db` for the database and `omero.cluster.read_only.repo` for the binary repository.

Read-only access to the database assumes that `INSERT` and `UPDATE` commands cannot be used over a JDBC connection. Read-only access to the binary repository assumes that filesystem changes cannot be made within any of the standard OMERO directories. In all cases OMERO’s *TempFileManager* expects write access to the volatile storage location that `OMERO_TMPDIR` can be used to configure.

For each of these configuration properties the server assumes read-write access by default or with a `false` setting. Set a property to `true` to have the server treat the corresponding resource as being read-only. Additionally, without a `true` setting the server may log a warning and regard a resource as being in read-only mode if it discovers that it does not have write access. The currently effective values are provided by the *configuration service* as `omero.cluster.read_only.runtime.db` and `omero.cluster.read_only.runtime.repo`.

Setting `omero.pixeldata.memoizer.dir.local` to a read-write directory allows a read-only server to create and use the Bio-Formats memo files that cache reader state. The server still checks the default `BioFormatsCache/` directory in the read-only binary repository for existing memo files that it can copy to this local directory.

```
$ omero config set omero.cluster.read_only.db true
$ omero config set omero.cluster.read_only.repo true
$ omero config set omero.pixeldata.memoizer.dir.local /tmp/BioFormatsCache
```

Note: If the deprecated configuration property `omero.cluster.read_only` is set to `true` then the server behaves as if all `omero.cluster.read_only.*` properties were set to `true` regardless of any other value that they have.

3.11.9 Collection counts

The `IContainer` interface has always provided a method for returning the count of some collection types via `getDetails().getCounts()`. The server has database views for all link collections. These are accessed through HQL directly, such as:

```
Long self = iAdmin.getEventContext().getCurrentUserId();
Image i = iQuery.findByQuery(
    "select i from Image i left outer join fetch i.annotationLinksCountPerOwner", null);
Map<Long, Long> countsPerOwner = i.getAnnotationLinksCountPerOwner();

// Map may be null if not fetched.
if (countsPerOwner != null) {

    // countOfAnnotationsForImageByUser
    Long count = countsPerOwner.get(self);
    if (count != null) {
        // do something
    }
}
```

Values written to the map will not be persisted to the database, since they are continually re-generated.

Pojo options

The `PojoOptions` configuration of what elements are counted has been removed from the API. Instead, the returned map contains all values for all users, and can be summed to acquire the total count.

Restrictions

Currently a Hibernate bug (waiting to be filed) prevents retrieving the counts on any other than the top-level object (“select this”).

Instructions

The views.sql script is automatically executed when initializing your database.

3.11.10 How To create a service

Overview

These instructions are for core developers only and may be slightly out of date. They will eventually be revised, but if you are looking for general instructions on extending OMERO with a service, see [Extending OMERO.server](#). If you would indeed like to create a core service, please contact the [forum](#).

To fulfill #306, r905 provides all the classes and modifications needed to create a new stateless service (where this varies from stateful services is also detailed). In brief, a service provider must create an [interface](#), an [implementation](#) of that interface, a [Spring configuration file](#), as well as modify the [server configuration](#) and the central [service factory](#) (These last two points stand to change with #314).

Note: With the creation of [OMERO.blitz](#), there are several other locations which need to be modified. These are also listed below.

Files to create

<https://github.com/ome/omero-common/blob/v5.6.4/src/main/java/ome/api/IConfig.java>

the interface which will be made available to client and server alike (which is why all interfaces must be located in the `/common` component). Only serializable and client-available types should enter or exit the API. Must subclass `ome.api.ServiceInterface`.

<https://github.com/ome/omero-server/blob/v5.6.10/src/main/java/ome/logic/ConfigImpl.java>

the implementation which will usually subclass `AbstractLevel{1,2}Service` or `AbstractBean` (See more below on **super-classes**) This class obviously requires the most work, both to fulfill the interface's contract and to provide all the metadata (annotations) necessary to properly deploy the service.

[https:](#)

[//github.com/ome/omero-server/blob/v5.6.10/src/main/resources/ome/services/service-ome.api.IConfig.xml](https://github.com/ome/omero-server/blob/v5.6.10/src/main/resources/ome/services/service-ome.api.IConfig.xml)

a [Spring](#) configuration file, which can “inject” any value available in the server (Omero) context into the implementation. Two short definitions are the minimum. (Currently not definable with annotations.) As explained in the file, the name of the file is not required and in fact the two definitions can be added to any of the files which fall within the lookup definition in the server's [beanRefContext.xml](#) file (see below).

<https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/java/ome/services/blitz/impl/ConfigI.java>

a [Ice](#) “servant” implementation which can use one of several methods for delegating to the `ome.api.IConfig` interface, but all of which support [throttling](#).

Files to edit (not strictly necessary, see #314)

<https://github.com/ome/omero-common/blob/v5.6.4/src/main/java/ome/system/ServiceFactory.java>
our central API factory, needs an additional method for looking up the new interface (**get<interface name>Service()**)

<https://github.com/ome/omero-server/blob/v5.6.10/src/main/resources/ome/services/serverSpring>
configurations, which makes the use of JNDI and JAAS significantly simpler.

<https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/slice/omero/API.ice> (**blitz**)
a ZeroC slice definition file, which provides cross-language mappings. Add the same service method to ServiceFactoryI as to ServiceFactory.java.

<https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/resources/ome/services/blitz-servantDefinitions.xml> (**blitz**)
a Spring configuration, which defines a mapping between Ice servants and Java services.

<https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/slice/omero/Constants.ice> (**blitz**)
a ZeroC slice definition file, which provides constants needed for looking up services, etc.

<https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/java/ome/services/blitz/impl/ServiceFactoryI.java> (**blitz**)
the central session in a blitz. Should always be edited parallel to ServiceFactory.java. Also optional in that MyServicePrxHelper.uncheckedCast(serviceFactoryI.getByName(String)) can be used instead.

Files involved

<https://github.com/ome/omero-server/blob/v5.6.10/src/main/resources/beanRefContext.xml>

<https://github.com/ome/omero-blitz/blob/v5.7.2/src/main/resources/beanRefContext.xml>
Singleton definitions which allow for the static location of the active context. These do not need to be edited, but in the case of the server `beanRefContext.xml`, it does define which files will be used to create the new context (of importance is the line `classpath*:ome/services/service-*.xml`). blitz's `beanRefContext.xml` defines the pattern `classpath*:ome/services/blitz-*.xml` to allow for blitz-specific configuration.

And do not forget the tests

<https://github.com/ome/omero-server/blob/v5.6.10/src/test/java/ome/server/itests/ConfigTest.java>
tests only the implementation without a container.

blitz: Currently, testing blitz is outside the scope of this document.

Things to be aware of

Local APIs

Several services implement a server-side subclass of the **ome.api** interface rather than the interface itself. These interfaces are typically in **ome.api.local**. Such local interfaces can provide methods that should not be made available to clients, but which are needed within the server. Though not currently used, the **@Local()** annotation on the implementation can list the local interface for future use. See [UpdateImpl](#) for an example.

Stateful services

Currently all stateful services are in their own component (`renderer` and `romio`) but their interface will still need to be under `common` for them to be accessible to clients. To be done.

3.11.11 OMERO sessions

OMERO sessions simplifies the handling of login sessions for *OMERO.blitz*.

In short:

- Sessions are a replacement for the standard JavaEE security infrastructure.
- Sessions unify the Blitz and RMI session handling, making working with Java RMI more like Blitz (since the JavaEE interaction is essentially “conversationless”).
- Sessions provide the ability (especially in Blitz) to quit a session and rejoin it later as long as it has not timed out, possibly useful for moving from one machine to another.
- Sessions provide the ability to share the same space. Two users/clients attached to the same session would experience the same life-cycle.
- Sessions provide a scratch space to which any data can be written for and by job/script executions.
- Sessions act as a global cache (in memory or on disk) to speed up various server tasks, including login. With further extensions like <http://terracotta.org/>, sessions could serve as a “distributed” cache.
- Sessions prevent sending passwords in plain text or any other form. After that, all session interactions take place via a shared secret key.

Design

All services other than `ISession`, assume that a user is logging in with a username equal to session uuid. Whereas previously one logged in with:

```
ome.system.Principal p = new ome.system.Principal("josh", "user", "User");
```

behind the scenes, now the “josh” value is replaced by the UUID of a `ome.model.meta.Session` instance.

The session is acquired by a call to:

```
ome.api.ISession.createSession(Principal principal, String credentials);
```

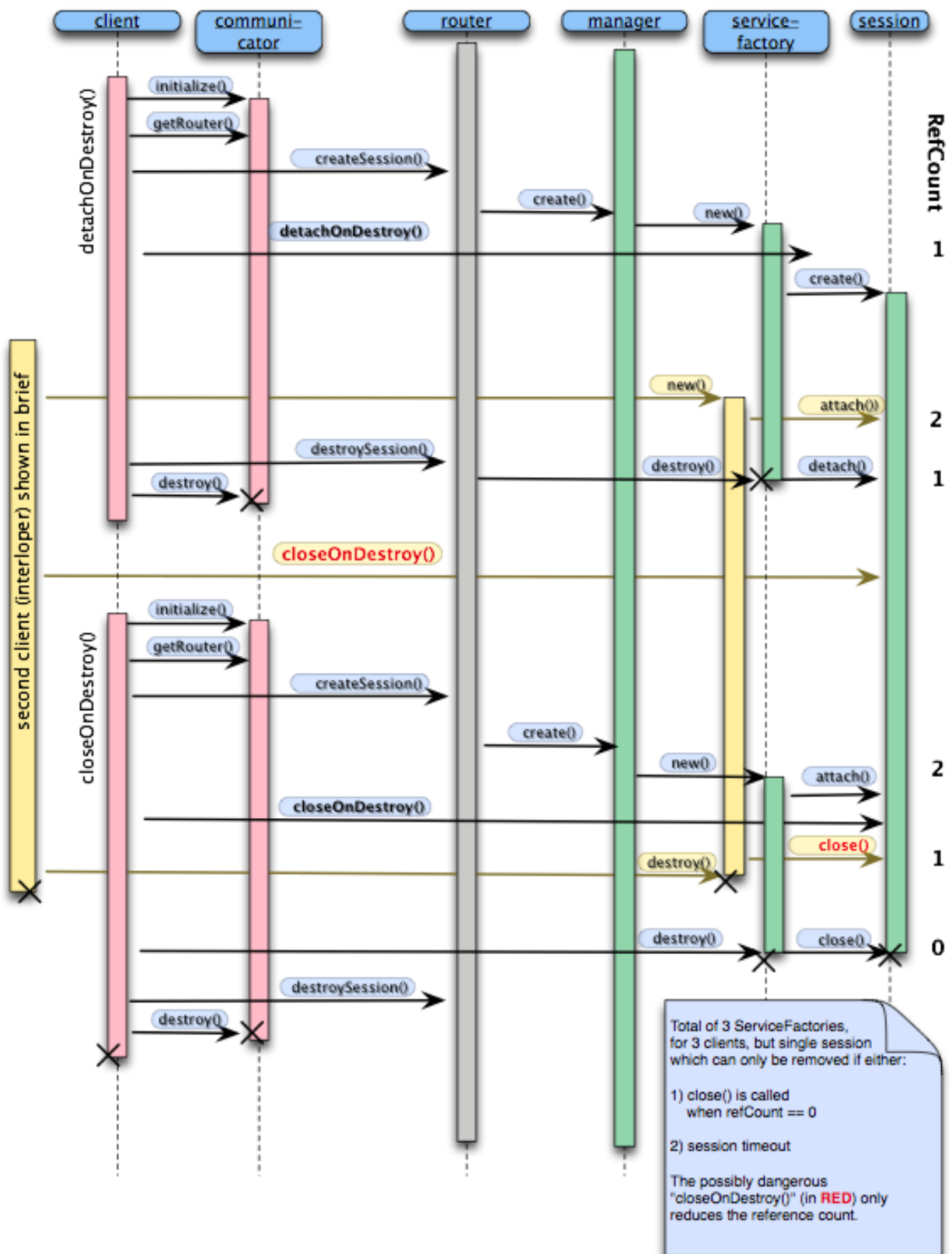
and carries information related to the current user’s session.

```
Session session;
session.getUuid();           // Unique identifier; functions as a temporary password. DO NOT
    ↪ SHARE IT.
session.getTimeToIdle(); // Number of milliseconds which the user can idle without
    ↪ session timeout
session.getTimeToLive(); // Total number of milliseconds for which the session can live
session.getStarted();     // Start of session
session.getClosed();      // if != null, then session is closed
```

These properties cannot be modified.

Other properties are for use by clients:

OMERO.blitz Session Creation and Destruction



```

session.getMessage();           // General purpose message statement
session.getAgent();             // Can be used to specify which program the user is
↪ using
session.getDefaultEventType();  // Default event type (the third argument "User" to
↪ Principal above)
session.getDefaultPermissions(); // String representation of umask (e.g. "rw----")

```

After changing a property on the session returned by `createSession()` it is possible to save them to the server via:

```
ome.api.ISession.updateSession(Session);
```

Finally, when finished, to conserve resources it is possible to destroy the session via:

```
ome.api.ISession.closeSession(Session);
```

Existing sessions

In *OMERO.blitz*, it is possible to reacquire the session if it is still active, by passing the previous session UUID as your password (User principal is ignored).

```

client = omero.client()
servicefactory = client.createSession()
iadmin = servicefactory.getAdminService()
olduuid = iadmin.getEventContext().sessionUuid

// lose connection

client = omero.client()
servicefactory = client.createSession(omero.sys.Principal(), olduuid)
// now reattached

```

See also:

Server security and firewalls

3.11.12 Aspect-oriented programming

Aspect-oriented programming is, among other things, the attempt to define and centralize cross-cutting concerns. In other words, it is not much more than the tried-and-true principle of modularization. Having possibly unseen aspects operating on a given class however, can complicate an initial examination of the code. Therefore, it is important to be aware of what portions of the OMERO code base are “advised” and where to find the advisors (in the case of OMERO solely interceptors).

In *Spring*, advisors are declared in the bean definition files (under <https://github.com/ome/omero-server/tree/v5.6.10/src/main/resources/ome/services>, `services.xml`, `hibernate.xml`, and others).

In these configuration files, various *Spring* beans (shared objects) are defined with names like “proxyHandler”, “eventHandler”, “serviceHandler”, and “transactionHandler”. Each of these is a method interceptor which is passed execution before the actual logic is reached. The interceptor can inspect or replace the return value, but can also stop the method execution from ever taking place.

Unlike with *AspectJ*, the AOP implementation used by OMERO only allows for the advising of interfaces. Simply creating a new service implementation via “new QueryImpl()” will not produce an advised object, which in turn will not function properly, if at all. Instead, advised objects can only be acquired from the *Spring context*.

By and large, only the *API service methods* are advised in OMERO.

Why?

Often, when implementing or adding code, it becomes clear just how many requirements are placed by libraries, the application server, and existing code on any new code. This can include transaction handling, session handling, security checks, object validation, logging etc. As a code-base grows, these dependencies slow development and make code unmanageable. AOP tries to reduce these dependencies by defining each of these concerns in a single place.

As a quick example, in OMERO transactions and exceptions are handled through method interceptors. Rather than writing:

```
void method1(){
    try {

        Transaction tx = new Transaction();
        tx.begin();
        // your code goes here
        tx.commit();
    } catch (TxException e) {
        tx.rollback();
    } catch (OtherException e) {

    }

}
```

you just write:

```
void method1(){
    // your code goes here
}
```

See also:

Aspect Oriented Programming

Chapter of the Spring documentation

AOP Alliance

Joint project defining interfaces for various AOP implementations

AspectJ

The arguable leader in Java/AOP development. Not used in Omero, but a good starting point.

Aspect-oriented programming

Wikipedia page on AOP

3.11.13 OmeroContext

The entire OMERO application (on a single JVM) resides in a single `ome.system.OmeroContext`. Each call belongs additionally to a single `org.hibernate.Session` (which can span over multiple calls) and to a single `ome.model.meta.Event` (which is restricted to a single task).

The container for all OMERO applications is the *OmeroContext* (<https://github.com/ome/omero-common/blob/v5.6.4/src/main/java/ome/system/OmeroContext.java>). Based on the Spring configuration backing the context, it can be one of `client`, `internal`, or `managed`. The use of a `ServiceFactory` simplifies this usage for the client.

Hibernate sessions

A `Hibernate Session` comprises a `Unit-of-Work` <https://www.martinfowler.com/eaCatalog/unitOfWork.html> which translates for OMERO's *OME-Remote Objects* model to a relational database. It keeps references to all Database-backed objects so that within a single session, object-identity stays constant and object changes can be persisted.

A session can span multiple calls by being disconnected from the underlying database transaction, and then reconnected to a new transaction on the next call (see <https://github.com/ome/omero-server/blob/v5.6.10/src/main/java/ome/tools/hibernate/SessionHandler.java> for the implementation).

For information about Events see *OMERO events and provenance*.

3.11.14 OMERO events and provenance

What is an event?

As described under *OmeroContext*, each method call takes place within a single application context (always the same), session, and event. Of these, only event is guaranteed to be unique for every task*. The <https://github.com/ome/omero-server/blob/v5.6.10/src/main/java/ome/security/basic/EventHandler.java> is responsible for creating new events.

Events as audit log

On each Database-update (INSERT/UPDATE/DELETE), an `EventLog` is created by a `HibernateInterceptor` which is then saved to the database at the end of the method call (in `UpdateImpl`).

Relationship to ModuleExecutions

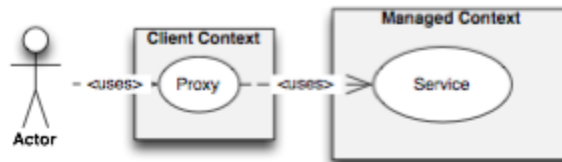
The OMERO Event plays a similar role to the `ModuleExecution` in the OME 2 system. They both contain time of create/update/deletion, status, and type information. Event, however, has lost its ACL/permissions role. These values have been moved to embedded values represented by the `Details` object. Event also is not linked to all the created `SemanticTypes` as was `ModuleExecution`, and so cannot fully represent the provenance data needed by the `AnalysisEngine`. At such time as the `AnalysisEngine` is ported to Java, the `ModuleExecution` object will have to be added.

* Here we say “task” and not method call, because all method calls to a single stateful service instance belong to the same event. This is the nature of a stateful service. Logically, however, it is a single action.

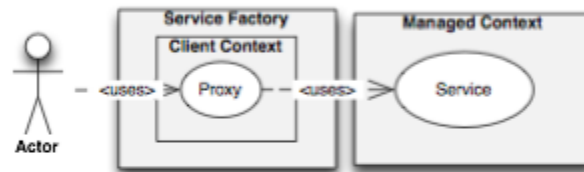
See also:

`Hibernate events` <https://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/events.html>

Omero Context Usage



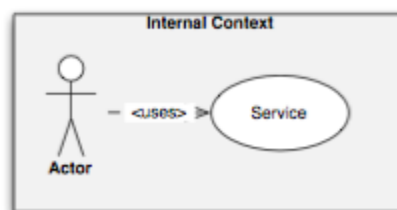
Client context: user accesses a proxy acquired from a context instantiated with `OmeroContext.getClientContext()`. All objects are serialized and a transaction starts at the barrier to the Managed Context (see below).



Client context (with ServiceFactory): more commonly, user instantiates a new `ServiceFactory()` and uses the public getters to acquire proxies.



Managed context: user accesses a wrapped service from a context instantiated with `OmeroContext.getManagedContext()`. Each call takes part in a single transaction.



Internal context: user accesses a raw service acquired from a context instantiated with `OmeroContext.getClientContext()`. No interception takes place. User must be careful to start transactions, open & flush sessions, and commit the transaction.

3.11.15 Properties

Under the `etc/` directory in both the source and the binary distributions, several files are provided which help to configure OMERO.server:

<https://github.com/ome/openmicroscopy/blob/develop/etc/omero.properties>

Since 5.5, the following repositories now have a `properties` file with the properties used in the repository itself. See `omero-model.properties`, `omero-common.properties`, `omero-server.properties`, `omero-blitz.properties`.

<https://github.com/ome/openmicroscopy/blob/develop/etc/hibernate.properties>

Required by Hibernate since some properties are only configurable via a `classpath:hibernate.properties` file

<https://github.com/ome/openmicroscopy/blob/develop/etc/logback.xml>

Logging configuration

<https://github.com/ome/openmicroscopy/blob/develop/etc/build.properties>

The properties that you will most likely want to change

etc/local.properties

Local file overriding `etc/build.properties` (used by build only)

The most useful of the properties are listed in a *glossary*.

On creation of an *OmeroContext*, the lookup for properties is (first wins):

- Properties passed into the constructor (if none, then the default properties in `config.xml`)
- System.properties set via “java -Dproperty=value”
- Configuration files in order listed.

This ordering is defined for the various components via “placeholder configurers” in the following file in <https://github.com/ome/omero-server>:

- <https://github.com/ome/omero-server/blob/v5.6.10/src/main/resources/ome/services/services.xml>

Once configured at start, all values declared in one of the mentioned ways can be used in Spring configurations via the syntax:

```
<bean id=...>
  <property name="mySetter" value="${property.name}"/>
</bean>
```

3.11.16 Using server queries internally

Overview

This page is aimed at internal server developers and does not contain information for how to perform API queries. If that is the kind of information you are looking for, you may find *OMERO Application Programming Interface* more useful as a starting point. OME’s *Hibernate 3.5 Training* additionally provides information on both API queries and server internals.

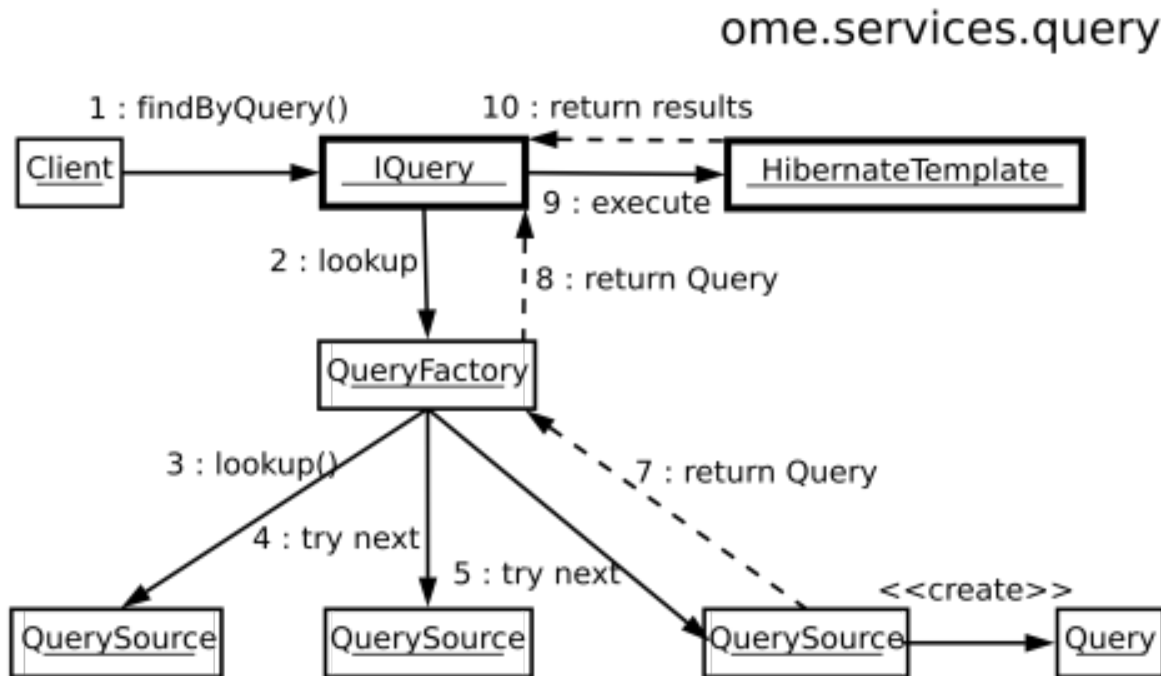


Fig. 14: omero.services.query

Introduction

The ome.services.queries package is intended to allow for the easy definition of queries by both developers and clients. Due to the fragility of HQL defined queries, a framework allowing for easy definition, multiple formats (Velocity templates, Database values, class files), and transparent lookup is critical.

Lookup happens among all QuerySources that are registered with the QueryFactory instance present in OMERO services. The first non-null Query instance returned by a QuerySource for a given String id is used.

Queries implement the HibernateCallback interface and are passed directly into an HibernateTemplate instance. Therefore, care should be taken as to which QuerySources are registered with the QueryFactory.

Parameters

Critical for using queries is the specification of named parameters, number of results to return, offset of the first result to return etc. These features are offered by the ome.parameters package. The ome.parameters.Parameters class is the starting point for building new parameters (although the ome.parameters.Filter object is used by some methods).

To specify parameters, instantiate a Parameters object either with or without a Filter object argument. The version with Filter object is useful for specifying the number of results to be returned and whether or not a java.util.Collection or a ome.model.IObject instance will be returned. For example,

```
Parameters p = new Parameters( new Filter().unique() );
```

will specify that the given query should return a single instance. An exception will be thrown if more than one result is found.

```
Parameters p = new Parameters( new Filter().unique().page(0,1) );
```


However, this will guarantee that only one result will be returned, since more than 1 result (“maxResults”) will be ignored. Here, an ordering of the results might make sense.

Once a Parameters instance is available, named parameters can be added using any of the `add...()` methods. These parameters will be dynamically bound during query preparation. For example, a query of the form:

```
select e from Experimenter e where omeName = :name
```

has one named parameter “name”, which can be specified by the call:

```
parameters.addString("name", "<myNameHere>");
```

Positional parameters of the form

```
select e from Experimenter where omeName = ?
```

are not supported.

Adding queries

Subclassing query

Other than by defining String queries via `new QueryDef()` TBD, the easiest way to create queries is to subclass `ome.services.query.Query`. The only non-optional requirements on the Query implementor are then to define the (possibly optional) named parameters to the Query, and to override the “buildQuery” (which must call one and only one of “setQuery()” or “setCriteria()”)

Other than that, the Query implementor can enable filters on the Hibernate session (an attempt is made to clean up after the Query runs), and in general use any of the Hibernate session methods.

Defining a QuerySource

A more involved but perhaps more rewarding method would be to implement `QuerySource` and configure `QueryFactory` to lookup query ids also in your `QuerySource`. This would allow you to write Velocity (or Freemarker/Ruby/Python/Groovy...) `QuerySources` which use some form of templating or scripting to generate HQL queries.

3.11.17 OMERO throttling

Throttling consists of reducing the total number of resources that one user or group can consume at a given time. The throttling service is a new component of *OMERO.blitz* which should ensure a more fair usage.

For example, each blitz server has a pre-defined maximum number of server threads. Any calls beyond this number must wait on a currently executing call to finish. Before throttling, a single user could consume all the available threads and all other users would have to wait.

With throttling, all invocations are placed on configurable on a **queue** which is worked on by any number of configurable **slots**. Each site can configure the number and type of slots based on which **throttling strategy** has been chosen.

Planning

Planned for milestone:3.0-Beta4, the infrastructure for throttling was committed to milestone <https://trac.openmicroscopy.org/ome/milestone/3.0-Beta3.1> with the in-thread strategy, which uses the calling thread for execution. This provides the same semantics as the current blitz server.

Other strategies include:

- a per-session strategy
- a per-user strategy
- a per-group strategy

each of which allows the session, user, or group a fair slice of execution, but no more. Within each strategy, the order of operation is guaranteed not to change once the execution reaches the server. However, there is nothing the server can do to prevent re-ordering if two calls are made by the client simultaneously.

More advanced strategies are possible based on total consumed resources over some window, or even a service-level agreement (SLA) or Quality of Service (QoS)-style planning. All strategies must guarantee a proper method ordering.

It is also intended that the throttling service provide limits to memory usage, database hits within a single transaction, and total execution time.

Terminology

- **Slots** - are the number of available executions that a single session, user, or group can perform simultaneously on a **single** machine. (If the server is clustered, there will be the given number of slots per hosts)
- **Hard** and **soft** limits - hard limits throw an `OverUsageException` and require some form of compensation on the clients. Soft limits, on the other hand, simply slow down, or throttle, execution to give other operations a chance to succeed.
- **Strictness** - when a strategy is configured as strict, then once a session, user, or group has reached its limits, the hard or soft limit will be enforced even if no one else is using the server. A non-strict policy will “borrow” someone else’s slot for the duration of one execution.

See also:

Scaling Omero

3.11.18 OMERO rendering engine

Description

The rendering component provides for the efficient rendering of raw pixels based on per-user display settings. A user can change settings and see them take effect in real time. Changes can also be persisted to the database and then viewed from another machine or even client.

Server-port

The rendering engine has been ported to also now sit on the server-side, though equally usable from any Java setting.

Optimizations

Here we have a listing of the various rendering engine optimizations that have taken place over time:

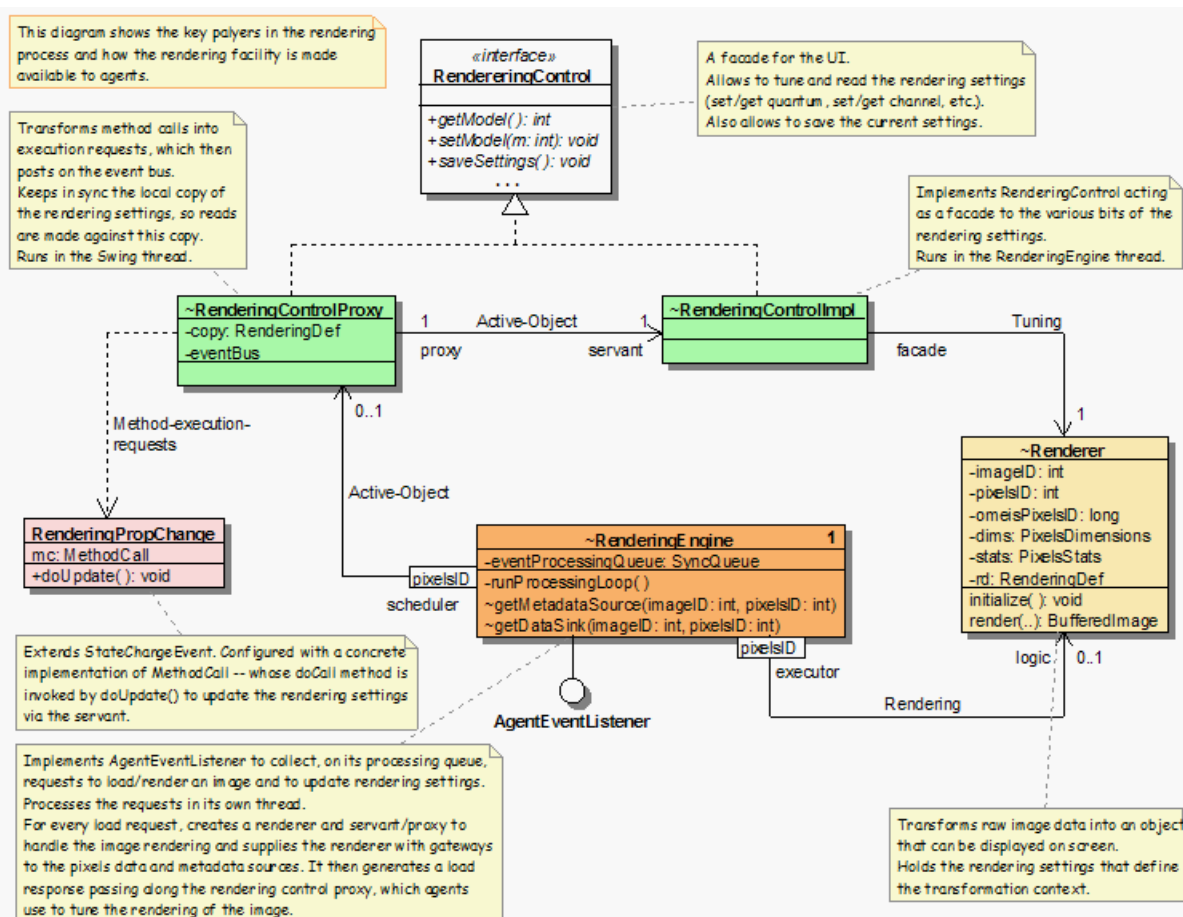
- Packed Integers (#449)
- Region Based Rendering (#450)
- Removal of RGB Rendering Model (#452)

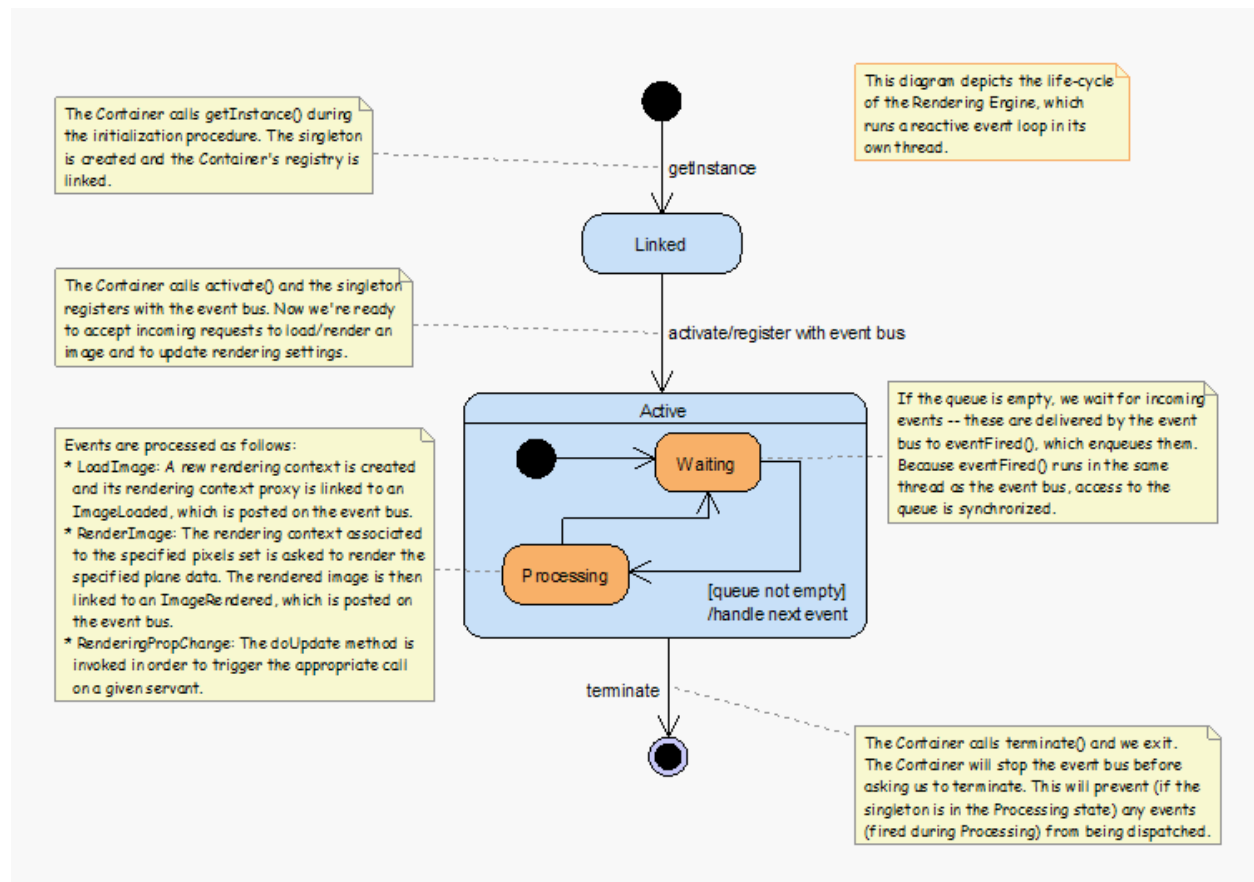
Compression

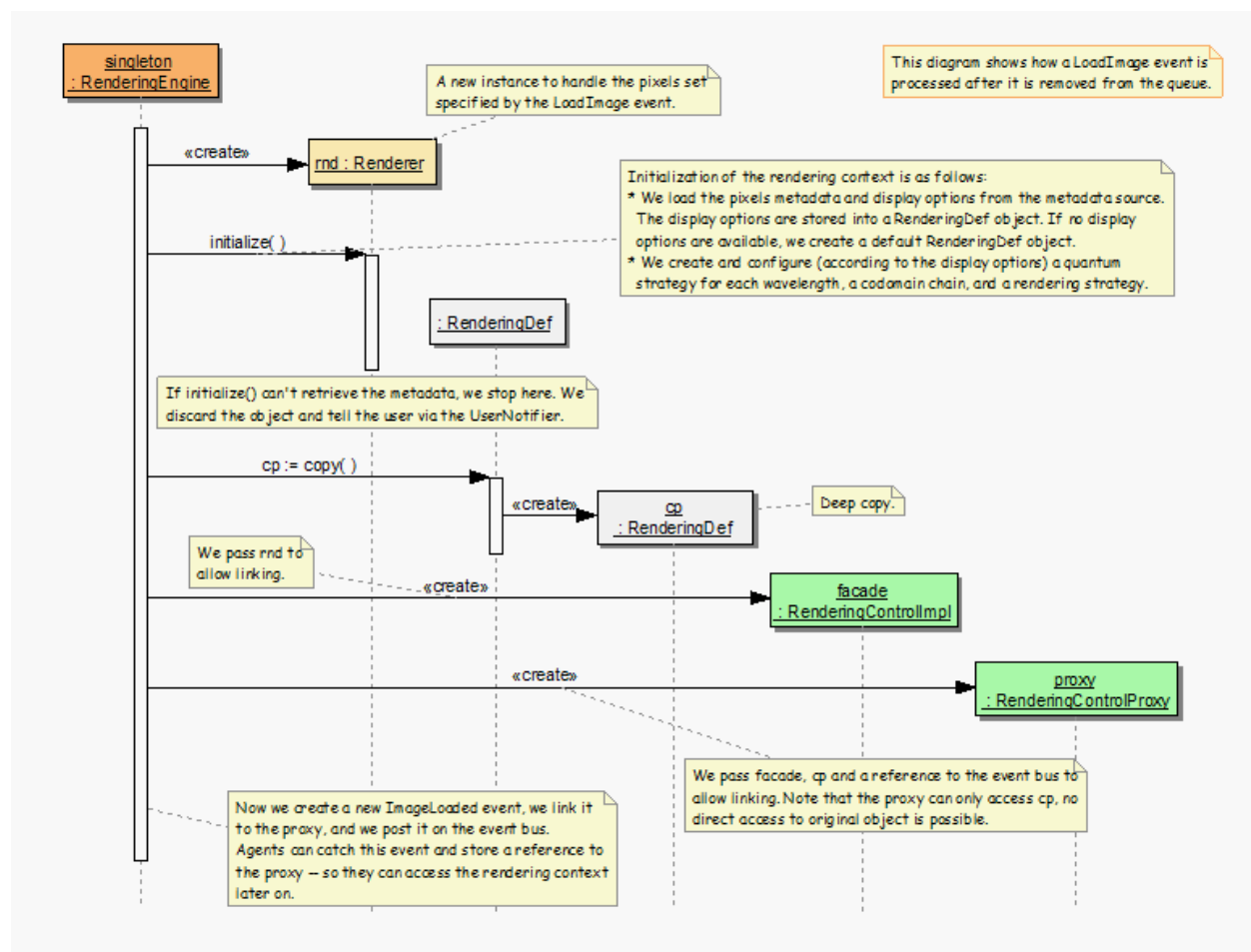
With r1744 and r1748, the rendering engine now supports compression. (#6)

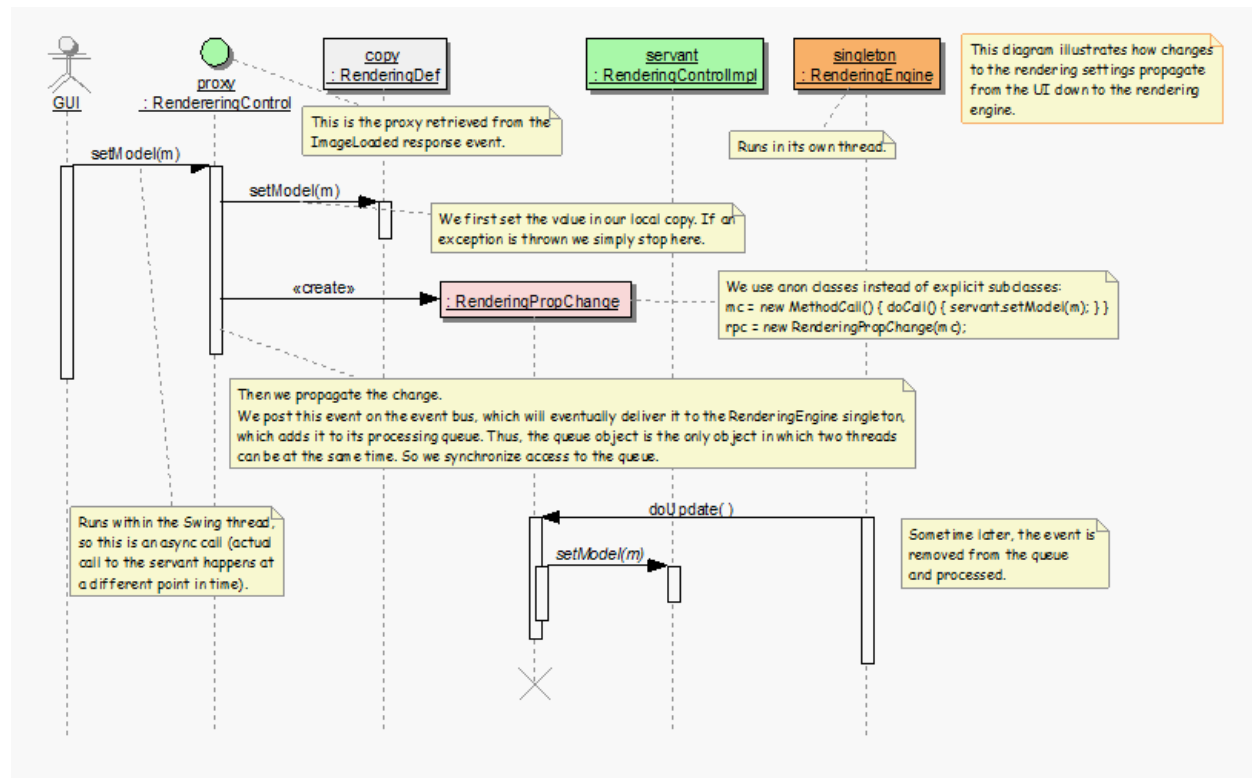
Design

The following diagrams describe the original design of the Rendering Engine. Designed initially for the client-side, much of this information needs to be updated. Textual explanations are included as notes in each diagram.

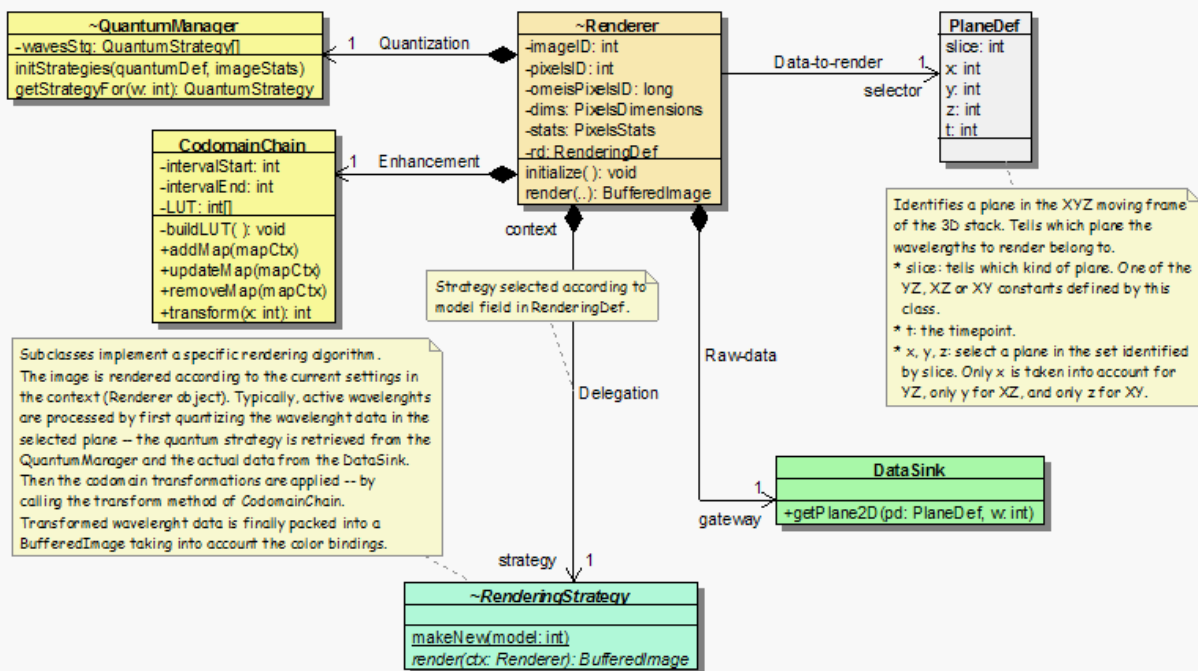


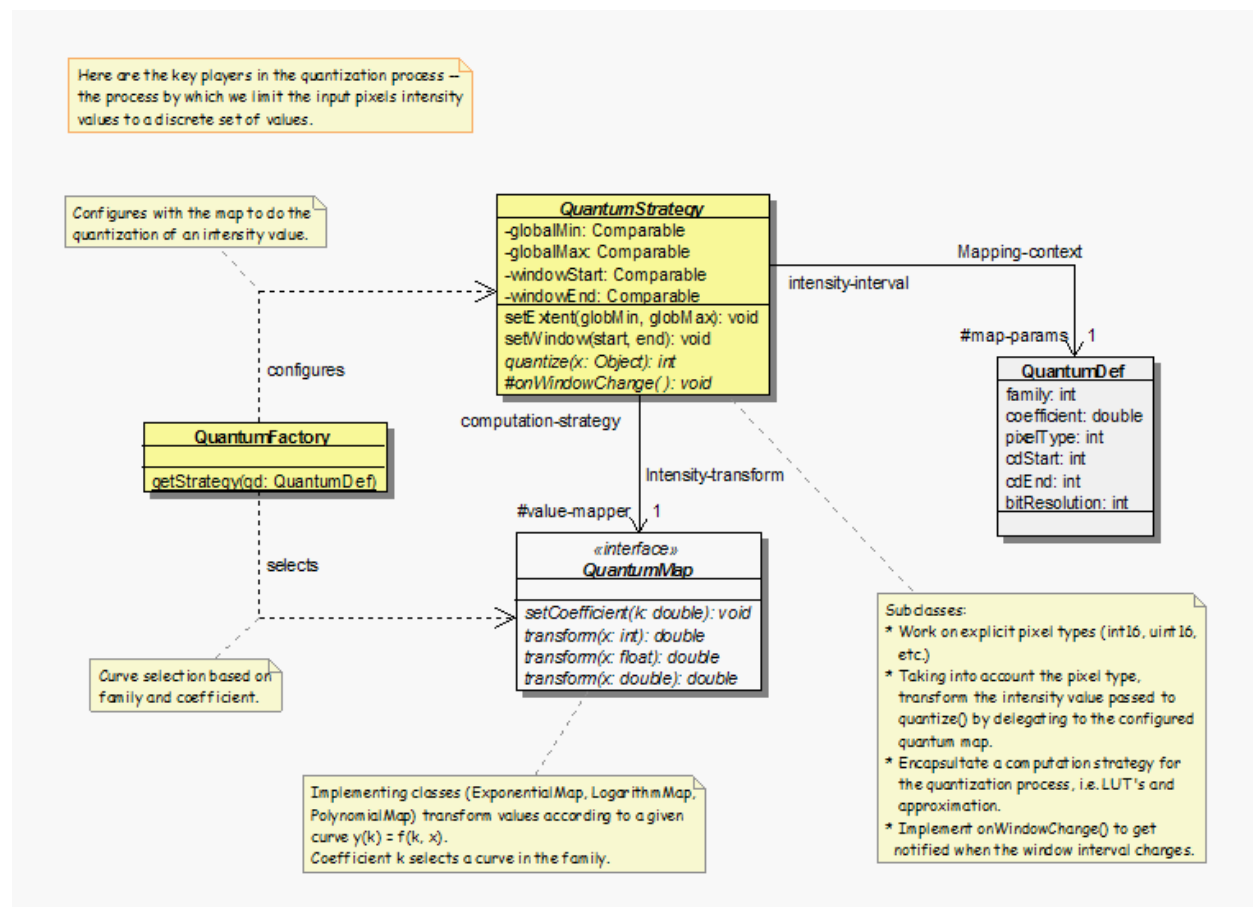


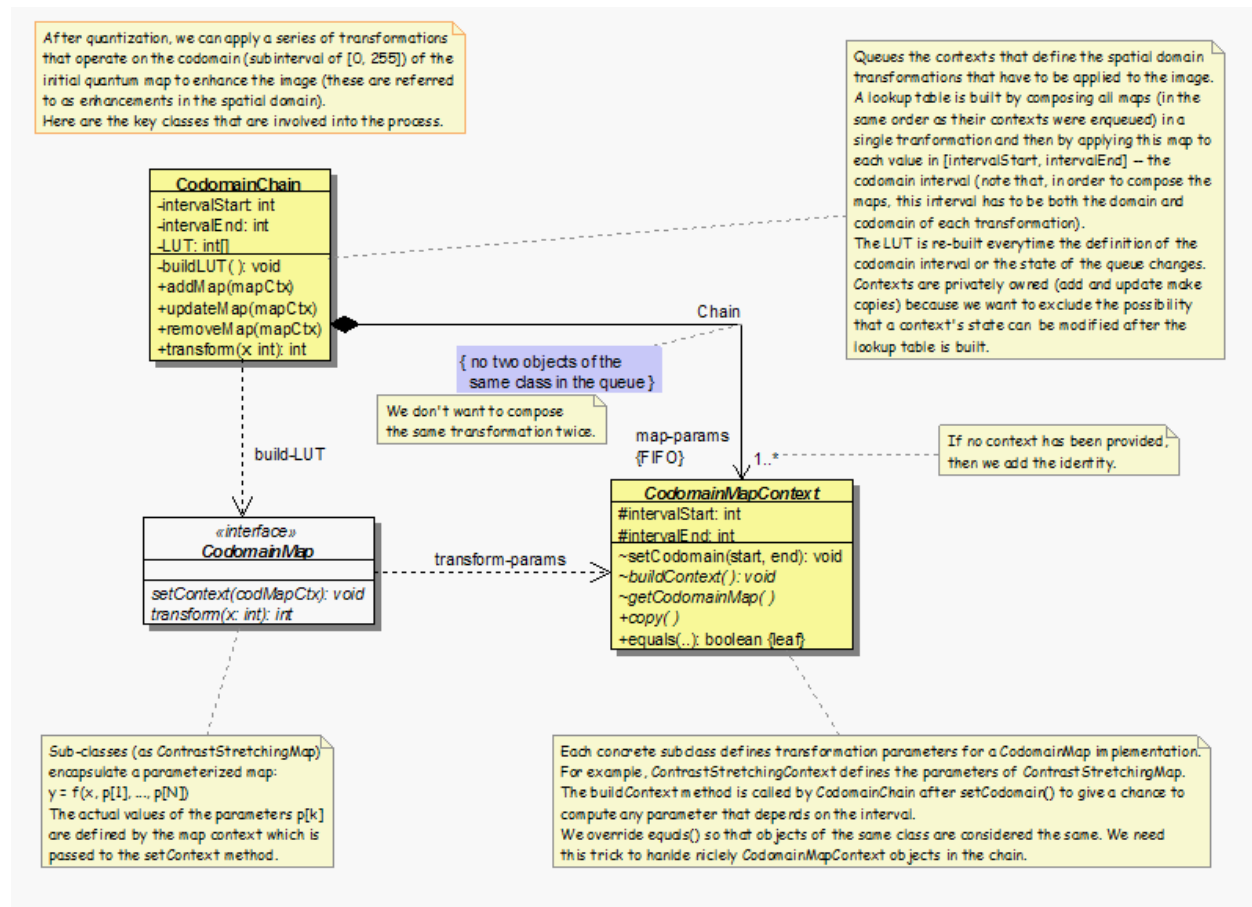


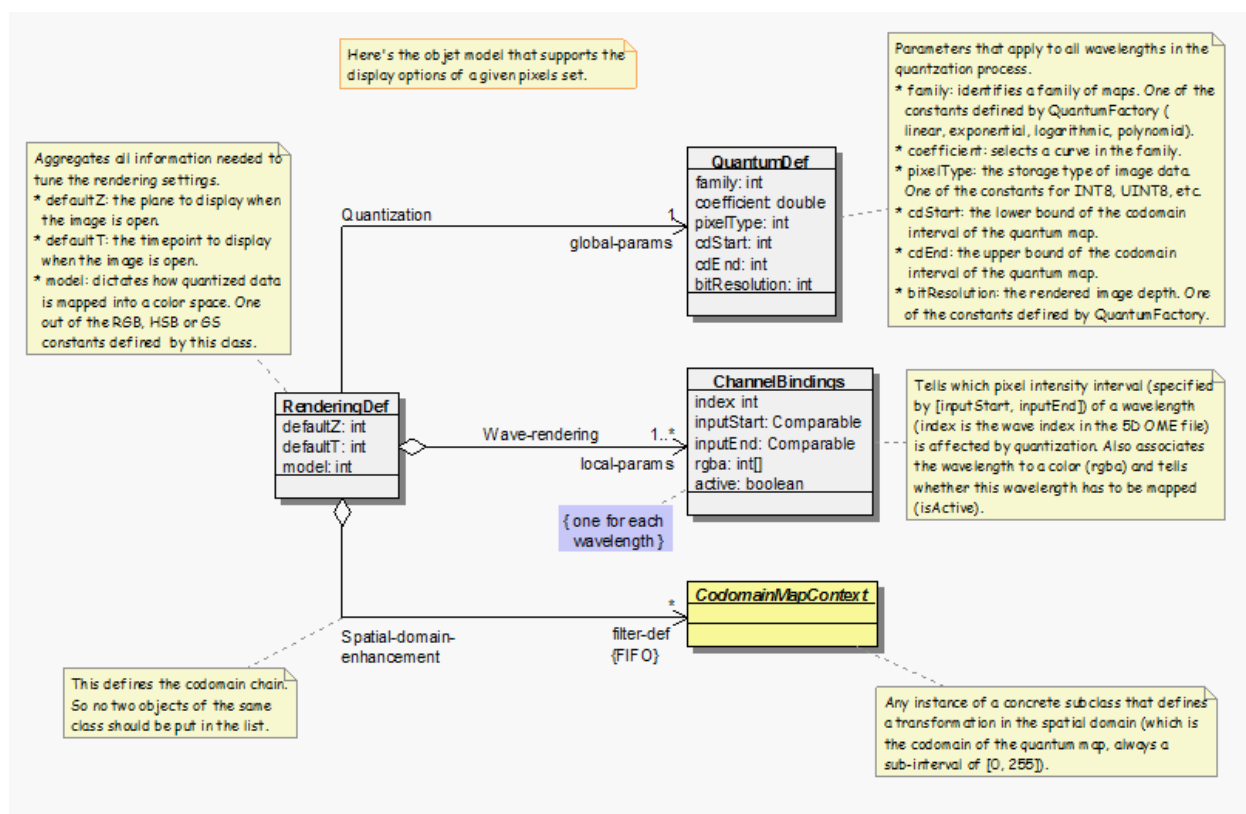


An OME image can aggregate more than one set of pixels – an example would be an OME image that is linked both to the original image data acquired from the microscope (this is one set of pixels) and the deconvolved data (yet another pixels set). In order to visualize a given pixels set within an image, we need both a rendering context to hold all sort of visualization options and algorithms that, taking into account the context information, transform the raw data into an image that can be displayed on screen.









3.11.19 Scaling Omero

There are several ways that OMERO, or any server system, can scale. Optimizing your system for more than one of these factors is non-trivial, but we try to lay out some guidelines below for what has worked, what almost certainly will not work, and what – under the right circumstances – might be optimal.

Concurrent invocations

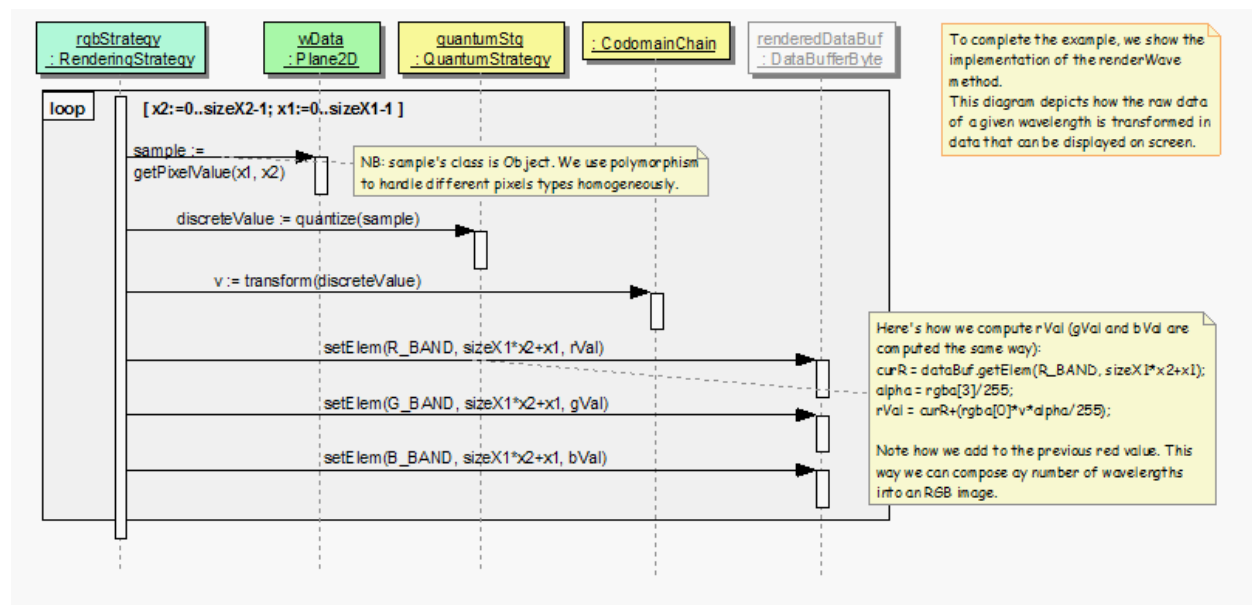
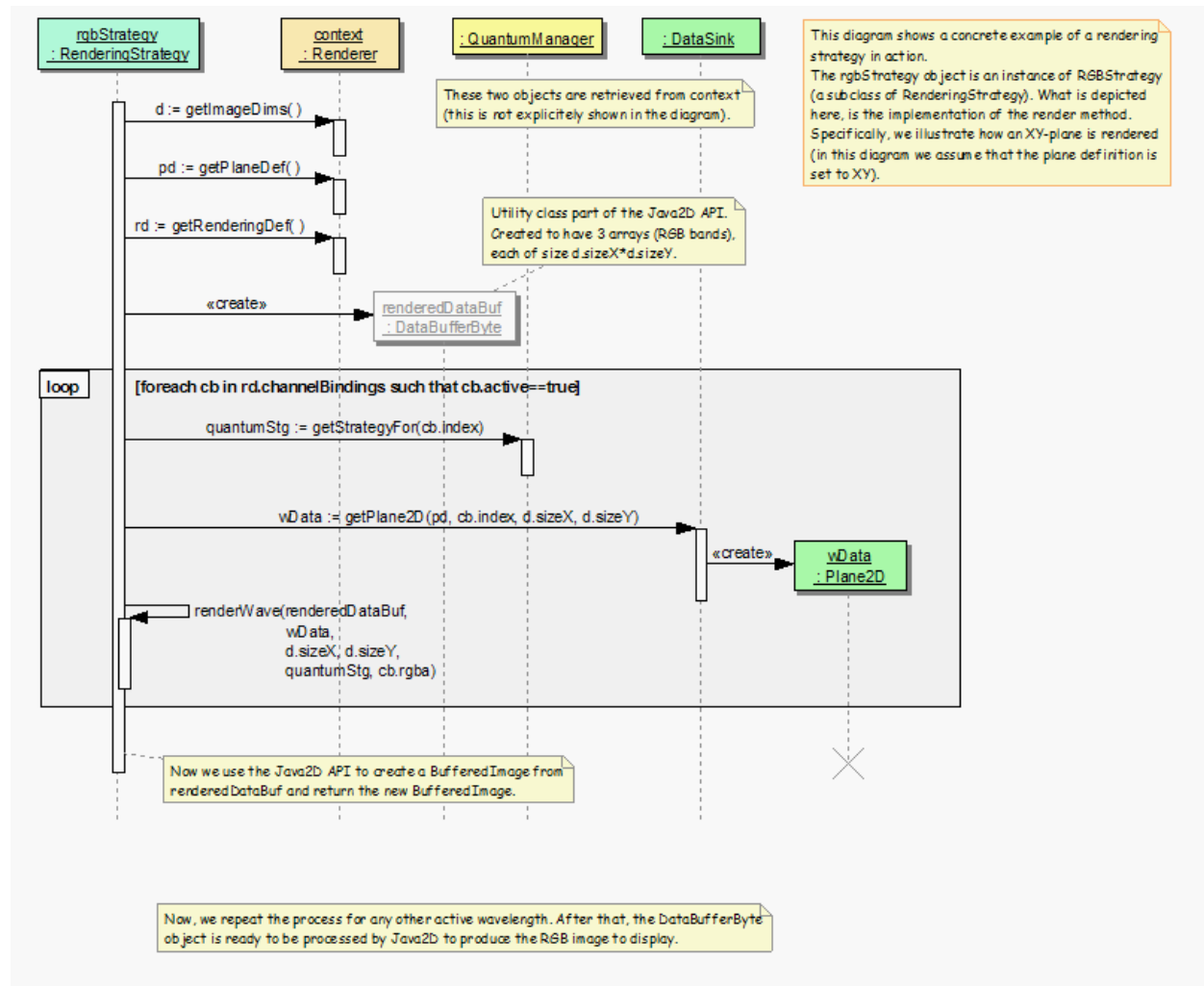
The bottlenecks for concurrent invocations are:

- database connections
- server threads
- the router

Database connections

Database servers, in general, have a maximum number of allowed connections. In postgres, the default `max_connections` is 100, though in many cases this will be significantly lower due to the available shared memory (SHMMAX). If OMERO were to use direct connections to the database, after `max_connections` invocations, all further attempts to connect to the server would fail with “too many connection” exceptions. Instead, OMERO uses a **connection pool** in front of Postgres, which manages many more simultaneous attempts to connect to the database. OMERO’s connection pool size as determined by `omero.db.poolsize` should be lower than `max_connections`.

With the default `max_connections` set to 64, it is possible to execute 500 queries simultaneously without database exceptions. Instead, one receives server exceptions.



Server threads

In *OMERO.blitz*, too many (500+ on the default configuration) simultaneous invocations will result in `ConnectionLost` exceptions. We are currently working on ways to extend the number of single invocations on one server, but a simpler solution is to start another *OMERO.blitz* server.

Total throughput

The bottlenecks for throughput are:

- maximum message size
 - server memory
 - IO
 - network
-

See also:

OMERO.server and PostgreSQL

Instructions about *OMERO.server* and PostgreSQL under UNIX and UNIX-like platforms.

OMERO.grid

Clustering

3.11.20 SqlAction

Internal server interface used to wrap all calls which speak JDBC directly. This allows special logic to be introduced where necessary for each RDBM.

Calls which use Hibernate for the cross-database conversion can use the `org.hibernate.Session` interface.

3.11.21 The server's view of administrator restrictions

OMERO 5.4 introduced the concept of a restricted administrator. These are generally more powerful than normal OMERO users and group owners but they do not have all the powers of full administrators such as `root`. A restricted administrator is the same as a full administrator except that *in specific ways* it is as if they are not an administrator at all. Those specific ways are exactly the restrictions listed in their user's `Experimenter.config` property. For that property, a name of `AdminPrivilege:Dance` (just an example!) with a value of `false` indicates that the restricted administrator's ability to dance is not that of a full administrator. Their ability to dance is instead that of a normal user who is not a member of the `system` group. A member of the `system` group with an empty `config` property is a full administrator without restrictions.

Restrictions

The `meta.ome.xml` mapping lists all the values of the `AdminPrivilege` enumeration. Each of those values corresponds to a kind of operation for which full administrators have greater privilege. If an administrator has any of the below restrictions then they do *not* have greater privilege for the corresponding operation. It instead becomes as if they were but a normal user in that respect.

Chgrp

move data to other groups

Chown

give data to other users

WriteOwned

create or edit OMERO model objects that have an owner and are not `OriginalFile` instances

WriteFile

create or edit `OriginalFile` objects that are in neither the managed repository nor the script repository

WriteManagedRepo

create or edit `OriginalFile` objects that are in the managed repository, which is to where imported image files are uploaded

WriteScriptRepo

create or edit `OriginalFile` objects that are in the script repository, which is where official scripts reside

DeleteOwned

delete OMERO model objects that have an owner and are not `OriginalFile` instances

DeleteFile

delete `OriginalFile` objects that are in neither the managed repository nor the script repository

DeleteManagedRepo

delete `OriginalFile` objects that are in the managed repository, which is to where imported image files are uploaded

DeleteScriptRepo

delete `OriginalFile` objects that are in the script repository, which is where official scripts reside

ModifyGroup

make changes to groups, such as their name or permissions level

ModifyGroupMembership

make changes to who is a member or owner of which group

ModifyUser

make changes to users, such as their name or institution

Sudo

become another user

ReadSession

read other users' session UUIDs

Bundling restrictions

It often makes sense to apply all but a certain bundle of restrictions to a user. For example, users working with other users' data may benefit from being able to go beyond their normal group restrictions only for the operations of `WriteOwned`, `WriteManagedRepo`, `WriteScriptRepo`, `Chgrp`. A facility manager importing on behalf of other users may appropriately be given all but the `Sudo` restriction. A human resources representative may be given all but the `ModifyGroup`, `ModifyGroupMembership`, `ModifyUser` restrictions. Much depends on how personnel roles are handled in each specific institution.

Warning: The `ReadSession` restriction should be applied to *all* restricted administrators. Without that restriction members of the `system` group can read other users' session UUIDs and join those sessions as the user. In contrast, while an administrator without the `Sudo` restriction can become other users, the security system prevents their using `sudo` to elevate their own administrative powers. An administrator cannot `sudo` to become `root` to escape their restrictions. A [security filter](#) assists in enforcing `ReadSession`.

Working with restrictions

Restricted administrators

Since OMERO 5.4 the `admin service` offers operations for managing restrictions on administrators. `createLightSystemUser` creates a new restricted administrator. `getAdminPrivileges` and `setAdminPrivileges` manage the restrictions on an existing administrator.

Using `setAdminPrivileges` to set an empty list of privileges fills that user's `Experimenter.config` property with a `false` value for every `AdminPrivilege` name. That user does not thus become like a normal user: they retain all administrative powers not explicitly restricted, such as being able to read all users' images.

Even for a normal user who is not a member of the `system` group and has no administrative powers, restrictions can still be set in their `Experimenter.config` property. Such restrictions have no effect while that user is not an administrator as they have no administrative powers to restrict.

`getCurrentAdminPrivileges` is useful for OMERO clients to find how the currently logged-in administrator is restricted. `getAdminsWithPrivileges` identifies the administrators who are sufficiently unrestricted in a given way.

Permissions on model objects

OMERO model objects have a `details property` that bears information on *object permissions*. In addition to the existing methods like `canEdit` and `canDelete`, the `canChgrp` and `canChown` methods were introduced in OMERO 5.4. Client software may find these permissions methods a useful guide as to what the current administrator may do to which objects.

Event context

Since OMERO 5.4 the `event context` for the current session, available from the admin service, has additional data members:

- `adminPrivileges` that lists the restrictions *not* applying to the current session. For non-administrators this list is empty as if they are wholly restricted. For restricted administrators it lists only the privileges that they enjoy. For full administrators all privileges are listed.
- `sudoerId`, `sudoerName` that for `sudo` sessions notes which administrator it was who became the current user.

Integration tests in Java

- `AdminServiceTest` tests the admin service operations for working with restrictions.
- `LightAdminPrivilegesTest` tests the restrictions from a security point of view: checking that applying even just one restriction to a user prevents all means of performing the corresponding operation.
- `LightAdminRolesTest` tests various user workflows: checking that with all but a given set of restrictions an administrator may perform useful sequences of operations.

Mapping of `OriginalFile.repo`

Since OMERO 5.4 the `repo` property of `OriginalFile` is mapped into the OMERO object model. Because the interpretation of an `OriginalFile` instance depends upon with which repository the file is associated, for security reasons the server greatly restricts the mutation of this property: users cannot simply switch a file from one repository to another.

The server must allow some setting of `repo`. It currently uses an indirect means of authenticating legitimately set values. Each running server has a secret key recorded in the `uuid` property of `Node`. This key is not available to OMERO clients, it is internal to the server. To set a new file's `repo` the `repository DAO` prefixes the file's name with the server's secret key. A database trigger recognizes this key from the `node` table, removes the prefix from the name, then allows the value of `repo` to be set.

Database triggers

While `BasicACLVoter` and `OmeroInterceptor` carry the bulk of the burden of enforcing restrictions on administrators, together with `AdminImpl` for the user and group management restrictions, the database system itself is also a key enforcement mechanism.

The `update service` is one means by which administrators may attempt to perform restricted operations. Hibernate's interceptors are not wholly suited to blocking exactly the prohibited actions so further barriers are built into the database that trigger upon specific data changes. The database must therefore have enough information to judge if an operation is permitted. OMERO 5.4 introduced two tables:

`_roles`

notes the server's configured IDs for special users and groups, such as `root` and `system` which are both usually `0`; set by `DBUserCheck` on server startup

`_current_admin_privileges`

notes the restrictions *not* applying to the current user on a per-transaction basis; maintained by `OmeroInterceptor` and frequently cleared by `LightAdminPrivilegesCleanup`

An example database trigger would be `user_config_delete_trigger` on the `experimenter_config` table. This trigger raises an exception if, for example, an `AdminPrivilege:Dance` name with a `false` value is to be removed from the `config` of a member of the `system` group by a user who themselves is restricted from dancing. This prevents the administrator whose dancing is restricted from lifting that restriction from another administrator so that they may be the one to newly dance.

3.11.22 Model graph operations

Overview

When the OMERO server acts on its model objects it must determine the impact on related objects. For instance, deleting an image may entail deleting users' rendering settings for that image, also the links to any datasets that the image is in. Understanding the details of the implementation substantially assists in debugging or creating server operations that act on the directed graph of model objects.

Motivation

The OMERO model objects are interlinked. Plates may have wells whose samples come from multiple runs. Both datasets and well samples may have images, but in different ways. Datasets, wells, images, among others, may all be annotated. Images themselves are not simple: for example, they may be in a fileset, they may have ROIs drawn on them, they may share an instrument with the projection of that image. All these entities are separate objects that can be thought of as forming the nodes (vertices) of a directed graph of relationships.

Various operations supported by the OMERO server, most commonly moving objects to a different group, or deleting them, may implicitly include many related objects. For example, if one deletes a fileset, one also deletes the images from that fileset, and even the comments on those images. This section describes how the graph of model objects is traversed and how the target set of related objects is determined.

This technical detail is important to understand if one wishes to,

- adjust the set of related model objects that are included in operations
- change the types of *OME-Remote Objects* model objects or the permissible links among them
- fix bugs in the related request objects defined in *Graphs.ice* that may be submitted to *OMERO sessions* for execution.

Approach

Graph node states and transitions

In determining which model objects to process, and how, each corresponding graph node is in one of these states:

adjective	rule format	Action enum	Orphan enum
irrelevant	[E]{i}	EXCLUDE	IRRELEVANT
relevant	[E]{r}	EXCLUDE	RELEVANT
orphaned	[E]{o}	EXCLUDE	IS_LAST
attached	[E]{a}	EXCLUDE	IS_NOT_LAST
to delete	[D]	DELETE	n/a
to include	[I]	INCLUDE	n/a
outside	[O]	OUTSIDE	n/a

“enum” refers to the enumerations defined in *GraphPolicy.java*. Note also that, as for the introduction to *Deleting in OMERO*, “links” are simply edges in the graph, distinct from the classes implementing *ILink.java* which themselves have several links, not least to their parent and child objects.

When traversal begins, the target objects are to be *included* (e.g., for *Chgrp2*) or *deleted* (e.g., for *Delete2*) and other objects are *irrelevant*.

A list of transition rules is associated with the requested operation. Each of the target objects is examined in turn and the rules matched against the state of that object and of those directly linked to it in either direction. If a rule matches, it may either abort the operation with an error condition or, more usually, change the state of any of the objects it matches. Changed objects are themselves queued for examination and rule matching. The traversal is complete when all queued objects have been examined with no further transition rule matches. Rules that can abort the operation are checked only after the other rules have completed processing. `GraphTraversal.java`'s `planOperation` method is at the heart of this matching process.

Further graph node states

Usual behavior is for *orphaned* objects related to the target objects to be included in the operation, but not the otherwise-*attached* objects, the non-orphans who have *excluded* parents that are to be neither deleted nor included. The related children that may be orphans are exactly those identified as being *relevant*. Transition rules match these against excluded parents to discover if the relevant objects do have any qualifying parents, changing them to be attached objects. If no further rules match and some objects remain as relevant, then they are automatically changed to orphans and examined for further rule matches. After that processing completes, attached objects are changed back to being relevant to confirm that excluded qualifying parents still exist to change them to being attached: this is necessary in case, after an object was considered attached, other rules changed all those qualifying parents from being excluded so that the object is now an orphan.

Objects that are changed to be *outside* are effectively rendered invisible, outside consideration in the execution phase. In the execution of an operation the graph traversal code removes links between included and excluded objects, but it allows links to remain between outside objects and other objects. Outside objects typically implement `IGlobal.java` and have no owner or group.

An additional aspect of objects' state is if permissions are to be checked for them. For instance, typically I may move only my own objects to a different group, but if another user tags my image with my tag, then I may still move my image and tag to a different group, also moving that link even though it is not my own object: in that case, permissions checking is disabled for that `ImageAnnotationLink`. All objects initially have permissions checking enabled, but the consequence of a rule may be to disable permissions checking, and if an object with permissions checking disabled matches a further rule, the objects changed by that rule also have permissions checking disabled.

Configuration

Defining the model graph transition rules

To reduce its complexity, `GraphTraversal.java` does not include specific detail of how to traverse the graph of *OME-Remote Objects* model objects. Instead, subclasses of `GraphPolicy.java` guide the traversal of the model object graph, configured by `blitz-graph-rules.xml` which names and defines the lists of transition rules. The named lists of rules are associated with request object classes by the definition of the `graphRequestFactory` bean in `blitz-servantDefinitions.xml`, which also specifies which model object properties may never be set to null in executing any requested operation.

`blitz-graph-rules.xml` begins with a comment that provides a key to the notation used for transition rules. The rules name and match model objects based on the state of the graph nodes, the types of the corresponding objects, the permissions the user has on those objects, and the names of the properties linking the objects. To illustrate this, the following sections briefly describe some different kinds of rule from the `deleteRules` list.

Propagating deletion

```
p:matches="L:ILink.parent = [D], L.child = C:[E]{o}/d"
p:changes="C:[D]"
```

If an ILink's parent (e.g., a dataset) is to be deleted, and its child (e.g., an image) is orphaned and deletable by the user, then delete the child also.

```
p:matches="PlateAcquisition[D].wellSample = WS:WellSample[E]"
p:changes="WS:[D]"
```

If a plate acquisition (run) is to be deleted, also delete its well samples (fields).

```
p:matches="Fileset[D] = I:Image[E].fileset"
p:changes="I:[D]"
```

If a fileset is to be deleted, then also delete its images.

```
p:matches="T:Thumbnail[E].pixels =/!o [D]"
p:changes="T:[D]/n"
```

If the pixels of a thumbnail are to be deleted, and are owned by a different user, then delete the thumbnail regardless of permissions on it.

Curtailing deletion

```
p:matches="Well[D].plate = C:[E]{!a}"
p:changes="C:{a}"
```

If a well is to be deleted but its plate is excluded and not attached, regard the plate as attached.

```
p:matches="C:Channel[E]{r}.pixels = Pixels[E]{i}"
p:changes="C:{a}"
```

If an irrelevant pixels object has a relevant channel, then regard the channel as attached.

```
p:matches="Pixels[D].relatedTo = P:[E]{!a}"
p:changes="P:{a}"
```

If a pixels object is to be deleted, regard any related, excluded pixels objects as attached. Because the pixels of an image are related to the pixels of a projection of that image, this rule prevents the deletion of an image from causing inadvertent deletion of the image's projections.

```
p:matches="L:ILink[!D].parent = [E]/d, L.child = C:[E]{r}"
p:changes="C:{a}"
```

If an ILink that is not to be deleted itself has a deletable, excluded parent and a relevant child, regard that child as attached.

Other kinds of transition rule

```
p:matches="E:IEnum[E]"
p:changes="E:[0]"
```

Regard excluded IEnum objects as being outside the operation. (Rules do not need to match on links among multiple objects.)

```
p:matches="F:Fileset[!D].images = [D], F.images = [!D]"
p:error="may not split {F}"
```

Throw an error if a fileset that is not to be deleted includes an image that is to be deleted and an image that is not to be deleted.

In reviewing the `chgrpRules` list, one sees conditions that require matching `$to_private` or `!$to_private`. A request, in this case `Chgrp2I.java`, may set arbitrary conditions upon which rules may be predicated. The `to_private` condition, or its absence, is used to cause different behavior when the objects are being moved into a private group.

Logging

Changing the log level

It is informative to observe the sequence of rule applications as the graph is traversed and decisions about model objects are made. To do so requires configuring *Omero logging* for the server, specifically <https://github.com/ome/openmicroscopy/blob/develop/etc/logback.xml>. To activate graph traversal debug logging, adjust the ends of the lines,

```
<logger name="omero.cmd.graphs" level="INFO"/>
<logger name="ome.services.graphs" level="INFO"/>
```

such that they instead read,

```
<logger name="omero.cmd.graphs" level="DEBUG"/>
<logger name="ome.services.graphs" level="DEBUG"/>
```

The resulting extra information in `var/log/Blitz-0.log` is of particular assistance in debugging: it pinpoints the rule applications that caused incorrect determinations of what action to take with model objects. Note that a `*` suffix on a model object referenced in the logs indicates that permissions are not to be checked for it.

Expanding the reports of transition rule matches

In the previous section, it can be seen that model objects that match rule conditions may be named. For example, in,

```
p:matches="Fileset[D] = I:Image[E].fileset"
p:changes="I:[D]"
```

the image is named I. When a rule matches, the debug logging reports which model object matched each name. If it remains unclear why a rule matched, further objects may be named. For example, changing the first line to name the fileset,

```
p:matches="F:Fileset[D] = I:Image[E].fileset"
```

would also report in the log which fileset matched the rule.

Encouragement

On first reading, the above may feel daunting. If model object graph traversal is not working as desired, thus requires adjustment, review of debug logs from `var/log/Blitz-0.log` typically pinpoints the cause and a minor adjustment to `blitz-graph-rules.xml` often suffices as the fix, with integration tests providing reassurance that the adjustment was acceptable. Sometimes it can take time and thought to devise that fix, but one can expect small changes to suffice to fix most bugs. In getting this new graph traversal implementation to initially pass integration testing, no test failures required a substantial rethink of the basic approach and `GraphTraversal.java` itself did not require a significant rewrite.

The actual lists of transition rules arose in part as a way to achieve the desired behavior and are not yet as simple and comprehensive as they could be. While they necessarily reflect the inherent complexity of the object model of *OME-Remote Objects*, there is potential for reviewing the rule lists and, perhaps with some additional marker interfaces, making them more succinct and regular. Incremental movements toward this goal are worth pursuing.

Options

Every one of the request object classes introduced in the new implementation of graph traversal is a derived class of `GraphModify2` and inherits data members that configure its operation. Each request may define additional data members for options specific to it, for instance `Chgrp2` requires the ID of the target group to be specified. The data members offered by all of the new requests are,

targetObjects

specifies which model objects the operation is to target

childOptions

specifies types of model objects (and, for annotations, namespaces) that should always or never be included in the operation (i.e. always considered to be orphans, or attached, regardless of excluded parents)

dryRun

specifies if the request is to determine which model objects would be included in and deleted by the operation, without actually executing the operation.

SkipHead

The `SkipHead` request allows specification of the target objects with reference to a common parent. It wraps an inner request data member that starts acting only after graph traversal reaches types listed in `startFrom`. For example, to target the images of a specific plate, give the plate in `targetObjects` and name `Image` in `startFrom`.

This feature is achieved by running the initial request with `dryRun` set to `true` and the graph traversal policy modified so as to not examine included nodes of types listed in `startFrom`. A subsequent request then runs, targeting the `startFrom` model objects that were included in the first request.

3.11.23 Java classes for model graph operations

This description of the roles played by server-side Java classes assumes familiarity with the *Model graph operations* machinery of `OMERO.server`.

Navigating the graph

When OMERO.server starts up the `GraphPathBean` reflects upon the object model and collates information about classes, subclasses, properties and their value types. This is what `GraphPathReport` uses to generate the *Glossary of all OMERO Model Objects*.

Traversing and acting

OMERO.server's `GraphTraversal` is at the core of all graph operations,

- querying the database to establish the model graph, with the help of `GraphPathBean`
- applying the graph operation's policy rules and changing the graph node states
- acting on the model objects according to the final state of the graph.

`GraphTraversal`'s `Processor` interface is implemented by specific graph requests to act on the selected model objects. `GraphTraversal` implements its `PlanExecutor` interface with code that calls those `Processor` methods: it provides that `PlanExecutor` implementation back to requests so that they can control exactly if or when to act via their `Processor` implementation.

`ModelObjectSequencer` ensures that objects are acted upon in the proper order. For example, in deleting `OriginalFile` instances, a directory's contents are deleted before their containing directory is deleted.

In OMERO.blitz, `BaseGraphTraversalProcessor` offers a useful base class for implementing `Processor` and `Null-GraphTraversalProcessor` has no effects at all. Several graph requests define their own `InternalProcessor` class.

Policy rules for node transitions

`GraphTraversal` manages the traversal of the model graph but it is instances of OMERO.server's `GraphPolicy` that decide how the graph's nodes are to change state during traversal. The class is instantiated by the static `parseRules` method of `GraphPolicyRule` which provides a `GraphPolicy` based on parsing a sequence of `GraphPolicyRule` instances. Each of those rules describes in textual form how it matches graph fragments and what to do in the event of a match.

OMERO.blitz's `BaseGraphPolicyAdjuster` provides convenient hooks for adjusting how an existing `GraphPolicy` transitions nodes. Classes that do such adjustment include,

`ChildOptionsPolicy`

marks certain nodes as `IS_LAST` or `IS_NOT_LAST` once they are `RELEVANT`

`SkipHeadPolicy`

1. in skipping the head, prevents traversal beyond certain node types
2. in processing the remaining graph, preserves permissions overrides established in the first phase

`SkipTailPolicy`

prevents traversal beyond certain node types

OMERO.server provides the `GraphPolicyRulePredicate` interface which is used for the `;` suffix notation in rule matches. For example, `GroupPredicate` can match `group=system` and `PermissionsPredicate` can match `perms=r?ra??`.

OMERO.blitz graph requests

The *Graph requests* of OMERO.blitz benefit from helper classes. `GraphRequestFactory` instantiates the graph request implementations and provides them means to create a context-aware `GraphHelper`. This helper includes the code that is common to many of the graph requests. Helper methods not requiring any context are instead collected in the stateless `GraphUtil`.

Symbols

\$OMERODIR, 199

-DCMAKE_CXX_FLAGS

cmake command line option, 445

-DCMAKE_EXE_LINKER_FLAGS

cmake command line option, 445

-DCMAKE_INCLUDE_PATH

cmake command line option, 444

-DCMAKE_LIBRARY_PATH

cmake command line option, 445

-DCMAKE_MODULE_LINKER_FLAGS

cmake command line option, 445

-DCMAKE_PREFIX_PATH

cmake command line option, 444

-DCMAKE_PROGRAM_PATH

cmake command line option, 445

-DCMAKE_SHARED_LINKER_FLAGS

cmake command line option, 445

-DCMAKE_VERBOSE_MAKEFILE

cmake command line option, 445

-DIce_DEBUG

cmake command line option, 445

-DIce_HOME

cmake command line option, 445

-DIce_INCLUDE_DIR

cmake command line option, 445

-DIce_SLICE2XXX_EXECUTABLE

cmake command line option, 445

-DIce_SLICE_DIR

cmake command line option, 445

-DIce_<C>_LIBRARIES

cmake command line option, 445

-T

omero-import command line option, 11

-U

omero-import command line option, 11

--advanced-help

omero-import command line option, 14

--all

omero-help command line option, 9

--archived

omero-fs-images command line option, 219

--bulk

omero-import command line option, 12

--cache

omero-fs-importtime command line option, 13

--check

omero-fs-sets command line option, 218

--debug

omero-import command line option, 14

--depth

omero-import command line option, 12

--description

omero-import command line option, 11

--dry-run

omero-chgrp command line option, 35

omero-chown command line option, 38

omero-delete command line option, 32

--email

omero-import command line option, 14

--errs

omero-import command line option, 11

--exclude

omero-chgrp command line option, 34

omero-chown command line option, 37

omero-delete command line option, 31

--extended

omero-fs-images command line option, 219

omero-fs-sets command line option, 218

--file

omero-import command line option, 11

--force

omero-delete command line option, 32

--force-rewrite

omero-admin-start command line option, 211

omero-admin-stop command line option, 211

--foreground

omero-admin-start command line option, 211

--group

omero-login command line option, 24

--groups

omero-fs-usage command line option, 222

--help

- omero-admin-start command line option, 211
- omero-admin-stop command line option, 211
- omero-fs-images command line option, 219
- omero-fs-mkdir command line option, 222
- omero-fs-rename command line option, 220
- omero-fs-repos command line option, 217
- omero-fs-sets command line option, 218
- omero-fs-usage command line option, 221
- omero-import command line option, 11
- pytest command line option, 373

--include

- omero-chgrp command line option, 34
- omero-chown command line option, 37
- omero-delete command line option, 30

--iterate

- omero-export command line option, 23

--java-help

- omero-import command line option, 14

--key

- omero-login command line option, 24

--limit

- omero-fs-images command line option, 219
- omero-fs-sets command line option, 218

--list

- omero-help command line option, 9

--logprefix

- omero-import command line option, 11

--logs

- omero-import command line option, 14

--managed

- omero-fs-repos command line option, 217

--markers

- pytest command line option, 373

--name

- omero-import command line option, 11

--no-move

- omero-fs-rename command line option, 220

--offset

- omero-fs-images command line option, 219
- omero-fs-sets command line option, 218

--order

- omero-fs-images command line option, 219
- omero-fs-sets command line option, 218

--ordered

- omero-chgrp command line option, 34
- omero-chown command line option, 37
- omero-delete command line option, 31

--output

- omero-import command line option, 11

--parallel-fileset

- omero-import command line option, 13

--parallel-upload

- omero-import command line option, 13

--parents

- omero-fs-mkdir command line option, 222

--password

- omero-login command line option, 24

--port

- omero-login command line option, 24

--recursive

- omero-help command line option, 9

--report

- omero-chgrp command line option, 35
- omero-chown command line option, 37
- omero-delete command line option, 31
- omero-fs-usage command line option, 222
- omero-import command line option, 14

--server

- omero-login command line option, 24

--size_only

- omero-fs-usage command line option, 222

--skip

- omero-import command line option, 12

--style

- omero-fs-images command line option, 219
- omero-fs-repos command line option, 217
- omero-fs-sets command line option, 218
- omero-fs-usage command line option, 221

--sudo

- omero-login command line option, 24

--summary

- omero-fs-importtime command line option, 13

--target

- omero-import command line option, 11

--units

- omero-fs-usage command line option, 222

--upload

- omero-import command line option, 14

--user

- omero-login command line option, 24

--wait

- omero-fs-usage command line option, 221

--with-transfer

- omero-fs-sets command line option, 218

--without-images

- omero-fs-sets command line option, 218

-d

- omero-import command line option, 11

-f

- omero-import command line option, 12

-g

- omero-import command line option, 11
- omero-login command line option, 24

-h

omero-admin-start command line option, 211
 omero-admin-stop command line option, 211
 omero-fs-images command line option, 219
 omero-fs-mkdir command line option, 222
 omero-fs-rename command line option, 220
 omero-fs-repos command line option, 217
 omero-fs-sets command line option, 218
 omero-fs-usage command line option, 221
 omero-import command line option, 11
 pytest command line option, 373

-k
 omero-login command line option, 24
 pytest command line option, 372

-m
 pytest command line option, 373

-n
 omero-import command line option, 11

-p
 omero-import command line option, 11
 omero-login command line option, 24

-r
 omero-import command line option, 11

-s
 omero-import command line option, 11
 omero-login command line option, 24
 pytest command line option, 373

-u
 omero-login command line option, 24

-w
 omero-login command line option, 24

-x
 omero-import command line option, 11

A

Action, 313
 Add Users to Groups, 320
 addData() (*omero.grid.Table* method), 479
 admin (*omero.query.timeout* property), 282
 Administrator, 309
 admins (*omero.web* property), 289
 Analyst, 320
 analyzer (*omero.search* property), 284
 Annotate, 313
 append (*omero.jvmcfg* property), 264
 application_server (*omero.web* property), 290
 apps (*omero.web* property), 290
 auth (*omero.mail.smtp* property), 270
 authority (*omero.db* property), 259

B

background_threads (*omero.threads* property), 274
 background_timeout (*omero.threads* property), 274
 backoff (*omero.pixeldata* property), 276

base (*omero.ldap* property), 266
 base_include_template (*omero.web* property), 291
 batch (*omero.pixeldata* property), 277
 batch (*omero.search* property), 284
 bean (*omero.mail* property), 269
 bean (*omero.metrics* property), 272
 bean (*omero.policy* property), 280
 binary_access (*omero.policy* property), 280
 bitand (*omero.security.filter* property), 287
 bridges (*omero.search* property), 284
 broken, 375

C

caches (*omero.web* property), 291
 cancel_timeout (*omero.threads* property), 274
 center_plugins (*omero.web.ui* property), 304
 ch.qos.logback.core.Appender.error, 243
 Change ownership, 313
 Chgrp, 320
 chmod_strategy (*omero.security* property), 287
 Chown, 320
 chunk_size (*omero.web* property), 291
 Ciphers (*omero.glacier2.IceSSL* property), 262
 class (*omero.mail.smtp.socketFactory* property), 271
 client_downloads_base (*omero.web.login* property), 294
 cmake command line option
 -DCMAKE_CXX_FLAGS, 445
 -DCMAKE_EXE_LINKER_FLAGS, 445
 -DCMAKE_INCLUDE_PATH, 444
 -DCMAKE_LIBRARY_PATH, 445
 -DCMAKE_MODULE_LINKER_FLAGS, 445
 -DCMAKE_PREFIX_PATH, 444
 -DCMAKE_PROGRAM_PATH, 445
 -DCMAKE_SHARED_LINKER_FLAGS, 445
 -DCMAKE_VERBOSE_MAKEFILE, 445
 -DIce_DEBUG, 445
 -DIce_HOME, 445
 -DIce_INCLUDE_DIR, 445
 -DIce_SLICE2XXX_EXECUTABLE, 445
 -DIce_SLICE_DIR, 445
 -DIce_<C>_LIBRARIES, 445
 CMAKE_INCLUDE_PATH, 444
 CMAKE_LIBRARY_PATH, 444
 columns (*omero.grid.Data* attribute), 478
 config (*omero.ldap* property), 266
 config (*omero.mail* property), 270
 connect_timeout (*omero.ldap* property), 266
 connection
 omero-login command line option, 23
 connectiontimeout (*omero.mail.smtp* property), 271
 cors_origin_allow_all (*omero.web* property), 291
 cors_origin_whitelist (*omero.web* property), 291
 Create and Edit Groups, 320

Create and Edit Users, [320](#)

cron (*omero.pixeldata property*), [277](#)

cron (*omero.scripts.cache property*), [283](#)

cron (*omero.search property*), [285](#)

csrf_cookie_httponly (*omero.web property*), [291](#)

csrf_cookie_samesite (*omero.web property*), [292](#)

csrf_cookie_secure (*omero.web property*), [292](#)

csrf_trusted_origins (*omero.web property*), [292](#)

CXX, [444](#)

CXXFLAGS, [444](#)

D

Data Viewer, [320](#)

databases (*omero.web property*), [292](#)

db (*omero.cluster.read_only property*), [263](#)

debug (*omero.mail.smtp property*), [271](#)

debug (*omero.web property*), [292](#)

default (*omero.pixeldata.backoff property*), [276](#)

default_group (*omero.web.search property*), [299](#)

default_user (*omero.web.search property*), [299](#)

Delete, [313](#)

Delete Data, [320](#)

description (*omero.client.ui.tree.orphans property*), [257](#)

dialect (*omero.db property*), [259](#)

dir (*omero.data property*), [254](#)

dir (*omero.managed property*), [255](#)

dir (*omero.pixeldata.memoizer property*), [278](#)

dispose (*omero.pixeldata property*), [277](#)

django_additional_settings (*omero.web property*), [292](#)

driver (*omero.db property*), [259](#)

DYLD_LIBRARY_PATH, [126](#), [448](#)

E

Edit, [314](#)

enable (*omero.mail.smtp.starttls property*), [271](#)

enabled (*omero.client.ui.menu.dropdown.colleagues property*), [256](#)

enabled (*omero.client.ui.menu.dropdown.everyone property*), [257](#)

enabled (*omero.client.ui.menu.dropdown.leaders property*), [257](#)

enabled (*omero.client.ui.tree.orphans property*), [257](#)

enabled (*omero.web.feedback.comment property*), [293](#)

enabled (*omero.web.feedback.error property*), [293](#)

enabled (*omero.web.public property*), [298](#)

enabled (*omero.web.public.cache property*), [297](#)

environment variable

\$OMERODIR, [199](#)

CMAKE_INCLUDE_PATH, [444](#)

CMAKE_LIBRARY_PATH, [444](#)

CXX, [444](#)

CXXFLAGS, [444](#)

DYLD_LIBRARY_PATH, [126](#), [448](#)

GTEST_ROOT, [444](#)

ICE_CONFIG, [369](#), [371](#), [372](#), [376](#), [424](#)

ICE_HOME, [443](#), [444](#)

JAVA_OPTS, [205](#)

LD_LIBRARY_PATH, [126](#), [448](#)

OMERO_CONFIG, [209](#)

OMERO_HOME, [92](#), [93](#), [97](#), [101](#), [105](#), [110](#), [120](#), [133](#), [140](#), [147](#), [154](#), [162](#)

OMERO_PASSWORD, [24](#)

OMERO_PREFIX, [125](#), [126](#)

OMERO_SESSION_DIR, [26](#)

OMERO_SESSIONDIR, [25](#), [26](#), [86](#)

OMERO_TMPDIR, [86](#), [126](#), [207](#), [587](#), [729](#)

OMERO_USERDIR, [25](#), [86](#)

OMERODIR, [8](#), [92](#), [93](#), [97](#), [101](#), [105](#), [110](#), [113](#), [118](#), [120](#), [126](#), [133](#), [140](#), [147](#), [154](#), [162](#), [192](#), [197](#), [372](#), [509](#)

PATH, [126](#), [197](#), [445](#)

PYTHONPATH, [126](#), [322](#), [404](#)

SLICEPATH, [357](#), [445](#)

VERBOSE, [444](#)

error (*omero.throttling.method_time property*), [275](#)

event_log_loader (*omero.pixeldata property*), [277](#)

event_log_loader (*omero.search property*), [285](#)

excludes (*omero.search property*), [285](#)

F

fallback (*omero.mail.smtp.socketFactory property*), [271](#)

favicon_url (*omero.web property*), [293](#)

feedback (*omero.qa property*), [282](#)

from (*omero.mail property*), [270](#)

G

get_only (*omero.web.public property*), [298](#)

getAllMetadata() (*omero.grid.Table method*), [480](#)

getHeaders() (*omero.grid.Table method*), [478](#)

getMetadata() (*omero.grid.Table method*), [480](#)

getNumberOfRows() (*omero.grid.Table method*), [478](#)

getWhereList() (*omero.grid.Table method*), [479](#)

graphite (*omero.metrics property*), [272](#)

Group and Data Organizer, [321](#)

Group member, [309](#)

Group owner, [309](#)

group_filter (*omero.ldap property*), [266](#)

group_mapping (*omero.ldap property*), [267](#)

GTEST_ROOT, [444](#)

H

heap_dump (*omero.jvmcfg property*), [264](#)

heap_size (*omero.jvmcfg property*), [265](#)

host (*omero.client.web property*), [258](#)

host (*omero.db property*), [259](#)

host (*omero.mail property*), 270
 host (*omero.web.application_server property*), 290
 html_meta_referrer (*omero.web property*), 293

I

ICE_CONFIG, 369, 371, 372, 376, 424
 ICE_HOME, 443, 444
 IceSSL (*omero.glacier2 property*), 262
 icetransports (*omero.client property*), 256
 idle_timeout (*omero.threads property*), 274
 Importer, 321
 include_actions (*omero.search property*), 285
 include_types (*omero.search property*), 286
 index_template (*omero.web property*), 293
 indexer (*omero.throttling.method_time.error property*), 275
 indexer (*omero.throttling.method_time.warn property*), 275
 initial_zoom_level (*omero.client.viewer property*), 258
 initialize() (*omero.grid.Table method*), 479
 interpolate_pixels (*omero.client.viewer property*), 258
 IPv6 (*Ice property*), 264

J

JAVA_OPTS, 205
 jvm.fileDescriptorCountRatio, 243
 jython (*omero.launcher property*), 282
 jython (*omero.process property*), 283

K

key (*omero.web.public.cache property*), 297
 keyStore (*omero.security property*), 288
 keyStorePassword (*omero.security property*), 288

L

label (*omero.client.ui.menu.dropdown.colleagues property*), 256
 label (*omero.client.ui.menu.dropdown.everyone property*), 257
 label (*omero.client.ui.menu.dropdown.leaders property*), 257
 lastModification (*omero.grid.Data attribute*), 478
 LD_LIBRARY_PATH, 126, 448
 local (*omero.pixeldata.memoizer.dir property*), 278
 locking_strategy (*omero.search property*), 286
 logdir (*omero.web property*), 294
 login_failure_throttle_count (*omero.security property*), 288
 login_failure_throttle_time (*omero.security property*), 288
 login_incorrect_credentials_text (*omero.web property*), 294

login_logo (*omero.web property*), 294
 login_redirect (*omero.web property*), 294
 login_view (*omero.web property*), 295

M

ManualStrategy, 240
 matlab (*omero.launcher property*), 282
 matlab (*omero.process property*), 283
 max_file_size (*omero.search property*), 286
 max_fileset_size (*omero.search property*), 286
 max_partition_size (*omero.search property*), 286
 max_plane_float_override (*omero.pixeldata property*), 277
 max_plane_height (*omero.pixeldata property*), 278
 max_plane_width (*omero.pixeldata property*), 278
 max_projection_bytes (*omero.pixeldata property*), 278
 max_requests (*omero.web.application_server property*), 290
 max_size (*omero.client.download_as property*), 256
 max_system_memory (*omero.jvmcfg property*), 265
 max_table_download_rows (*omero.web property*), 295
 max_threads (*omero.threads property*), 274
 max_user_time_to_idle (*omero.sessions property*), 273
 max_user_time_to_live (*omero.sessions property*), 273
 maximum (*omero.sessions property*), 273
 maximum_multifile_download_size (*omero.web property*), 295
 maxpixels (*omero.pixeldata.backoff property*), 276
 memoizer_wait (*omero.pixeldata property*), 279
 merge_factor (*omero.search property*), 287
 metadata_panes (*omero.web.ui property*), 304
 middleware (*omero.web property*), 295
 min_system_memory (*omero.jvmcfg property*), 265
 min_threads (*omero.threads property*), 275
 Mix data, 314
 Move between groups, 314

N

name (*omero property*), 273
 name (*omero.client.ui.tree.orphans property*), 258
 name (*omero.db property*), 259
 new_user_group (*omero ldap property*), 267
 new_user_group_owner (*omero ldap property*), 267
 nginx_server_extra_config (*omero.web property*), 295
 nodedescriptors (*omero.server property*), 289

O

objects_read_interval (*omero.throttling property*), 276

objects_written_interval (*omero.throttling property*), 276
 ome.io.nio.PixelsService.minmaxTimes, 243
 ome.io.nio.PixelsService.tileTimes, 243
 ome.services.eventlogs.EventLogQueue.priorityCount, 243
 omero.grid.BoolColumn (*built-in class*), 477
 omero.grid.Column (*built-in class*), 476
 omero.grid.Data (*built-in class*), 478
 omero.grid.DoubleArrayColumn (*built-in class*), 477
 omero.grid.DoubleColumn (*built-in class*), 477
 omero.grid.FileColumn (*built-in class*), 476
 omero.grid.FloatArrayColumn (*built-in class*), 477
 omero.grid.ImageColumn (*built-in class*), 476
 omero.grid.LongArrayColumn (*built-in class*), 477
 omero.grid.LongColumn (*built-in class*), 477
 omero.grid.PlateColumn (*built-in class*), 476
 omero.grid.RoiColumn (*built-in class*), 476
 omero.grid.StringColumn (*built-in class*), 477
 omero.grid.Table (*built-in class*), 478
 omero.grid.Tables (*built-in class*), 476
 omero.grid.WellColumn (*built-in class*), 476
 OMERO_CONFIG, 209
 OMERO_HOME, 92, 93, 97, 101, 105, 110, 120, 133, 140, 147, 154, 162
 OMERO_PASSWORD, 24
 OMERO_PREFIX, 125, 126
 OMERO_SESSION_DIR, 26
 OMERO_SESSIONDIR, 25, 26, 86
 OMERO_TMPDIR, 86, 126, 207, 587, 729
 OMERO_USERDIR, 25, 86
 omero-admin-start command line option
 --force-rewrite, 211
 --foreground, 211
 --help, 211
 -h, 211
 omero-admin-stop command line option
 --force-rewrite, 211
 --help, 211
 -h, 211
 omero-chgrp command line option
 --dry-run, 35
 --exclude, 34
 --include, 34
 --ordered, 34
 --report, 35
 omero-chown command line option
 --dry-run, 38
 --exclude, 37
 --include, 37
 --ordered, 37
 --report, 37
 omero-delete command line option
 --dry-run, 32
 --exclude, 31
 --force, 32
 --include, 30
 --ordered, 31
 --report, 31
 omero-export command line option
 --iterate, 23
 omero-fs-images command line option
 --archived, 219
 --extended, 219
 --help, 219
 --limit, 219
 --offset, 219
 --order, 219
 --style, 219
 -h, 219
 omero-fs-importtime command line option
 --cache, 13
 --summary, 13
 omero-fs-mkdir command line option
 --help, 222
 --parents, 222
 -h, 222
 omero-fs-rename command line option
 --help, 220
 --no-move, 220
 -h, 220
 omero-fs-repos command line option
 --help, 217
 --managed, 217
 --style, 217
 -h, 217
 omero-fs-sets command line option
 --check, 218
 --extended, 218
 --help, 218
 --limit, 218
 --offset, 218
 --order, 218
 --style, 218
 --with-transfer, 218
 --without-images, 218
 -h, 218
 omero-fs-usage command line option
 --groups, 222
 --help, 221
 --report, 222
 --size_only, 222
 --style, 221
 --units, 222
 --wait, 221
 -h, 221
 omero-help command line option
 --all, 9

--list, 9
 --recursive, 9
 omero-import command line option
 -T, 11
 -U, 11
 --advanced-help, 14
 --bulk, 12
 --debug, 14
 --depth, 12
 --description, 11
 --email, 14
 --errs, 11
 --file, 11
 --help, 11
 --java-help, 14
 --logprefix, 11
 --logs, 14
 --name, 11
 --output, 11
 --parallel-fileset, 13
 --parallel-upload, 13
 --report, 14
 --skip, 12
 --target, 11
 --upload, 14
 -d, 11
 -f, 12
 -g, 11
 -h, 11
 -n, 11
 -p, 11
 -r, 11
 -s, 11
 -x, 11
 omero-login command line option
 --group, 24
 --key, 24
 --password, 24
 --port, 24
 --server, 24
 --sudo, 24
 --user, 24
 -g, 24
 -k, 24
 -p, 24
 -s, 24
 -u, 24
 -w, 24
 connection, 23
 OMERODIR, 8, 92, 93, 97, 101, 105, 110, 113, 118, 120, 126, 133, 140, 147, 154, 162, 192, 197, 372, 509
 open_with (omero.web property), 296
 .opengraph (omero.web.sharing property), 302

P

page_size (omero.web property), 296
 pass (omero.db property), 260
 password (omero.ldap property), 268
 password (omero.mail property), 270
 password (omero.web.public property), 298
 password_provider (omero.security property), 288
 password_required (omero.security property), 288
 patch (omero.db property), 260
 PATH, 126, 197, 445
 path (omero.fs.repo property), 254
 path_rules (omero.fs.repo property), 255
 percent (omero.jvmcfg property), 265
 PercentStrategy, 240
 perm_gen (omero.jvmcfg property), 265
 ping_interval (omero.web property), 296
 pipeline_css_compressor (omero.web property), 296
 pipeline_js_compressor (omero.web property), 296
 pipeline_staticfile_storage (omero.web property), 297
 plate_layout (omero.web property), 297
 poolsize (omero.db property), 260
 port (omero.db property), 260
 port (omero.mail property), 270
 port (omero.mail.smtp.socketFactory property), 271
 port (omero.web.application_server property), 290
 prefix (omero.ports property), 281
 prefix (omero.web property), 297
 prepared_statement_cache_size (omero.db property), 260
 Private, 310
 profile (omero.db property), 261
 properties (omero.db property), 261
 protocol (omero.mail.transport property), 272
 Protocols (omero.glacier2.IceSSL property), 263
 ProtocolVersionMax (omero.glacier2.IceSSL property), 262
 pytest command line option
 --help, 373
 --markers, 373
 -h, 373
 -k, 372
 -m, 373
 -s, 373
 python (omero.launcher property), 283
 python (omero.process property), 283
 PYTHONPATH, 126, 322, 404

R

ram_buffer_size (omero.search property), 287
 read() (omero.grid.Table method), 478
 read_only (omero.cluster property), 263
 read_timeout (omero.ldap property), 268
 Read-annotate, 310

Read-only, [310](#)
 Read-write, [310](#)
[readCoordinates\(\)](#) (*omero.grid.Table* method), [478](#)
[redirect_allowed_hosts](#) (*omero.web* property), [299](#)
[redirector](#) (*omero.cluster* property), [264](#)
[referral](#) (*omero.ldap* property), [268](#)
[registry](#) (*omero.ports* property), [281](#)
[registry_timeout](#) (*omero.grid* property), [264](#)
 Remove annotations, [314](#)
 Render, [314](#)
[repetitions](#) (*omero.pixeldata* property), [279](#)
[repetitions](#) (*omero.search* property), [287](#)
[repo](#) (*omero.cluster.read_only* property), [263](#)
[reporting_loops](#) (*omero.search* property), [287](#)
 Restricted Administrators, [309](#)
[right_plugins](#) (*omero.web.ui* property), [304](#)
[roi_limit](#) (*omero.client.viewer* property), [258](#)
[root_application](#) (*omero.web* property), [299](#)
[rowNumbers](#) (*omero.grid.Data* attribute), [478](#)

S

[scripts_to_ignore](#) (*omero.client* property), [256](#)
[secret_key](#) (*omero.web* property), [299](#)
[secure](#) (*omero.web* property), [299](#)
[secure_proxy_ssl_header](#) (*omero.web* property), [300](#)
[servants_per_session](#) (*omero.throttling* property), [276](#)
[server_id](#) (*omero.web.public* property), [298](#)
[server_list](#) (*omero.web* property), [300](#)
[session_cookie_age](#) (*omero.web* property), [300](#)
[session_cookie_domain](#) (*omero.web* property), [300](#)
[session_cookie_name](#) (*omero.web* property), [300](#)
[session_cookie_path](#) (*omero.web* property), [300](#)
[session_cookie_samesite](#) (*omero.web* property), [301](#)
[session_cookie_secure](#) (*omero.web* property), [301](#)
[session_engine](#) (*omero.web* property), [301](#)
[session_expire_at_browser_close](#) (*omero.web* property), [301](#)
[session_serializer](#) (*omero.web* property), [301](#)
[setAllMetadata\(\)](#) (*omero.grid.Table* method), [480](#)
[setMetadata\(\)](#) (*omero.grid.Table* method), [479](#)
[show_client_downloads](#) (*omero.web.login* property), [294](#)
[show_forgot_password](#) (*omero.web* property), [302](#)
[slf4j_minutes](#) (*omero.metrics* property), [272](#)
[slice\(\)](#) (*omero.grid.Table* method), [479](#)
[SLICEPATH](#), [357](#), [445](#)
[spec](#) (*omero.scripts.cache* property), [284](#)
[sql_action_class](#) (*omero.db* property), [261](#)
[ssl](#) (*omero.ports* property), [281](#)
[static_root](#) (*omero.web* property), [302](#)
[static_url](#) (*omero.web* property), [302](#)
[staticfile_dirs](#) (*omero.web* property), [303](#)
[statistics](#) (*omero.db* property), [261](#)

[strategy](#) (*omero.jvmcfg* property), [265](#)
 Sudo, [321](#)
[supported](#) (*omero.checksum* property), [254](#)
[sync_force](#) (*omero.sessions* property), [273](#)
[sync_interval](#) (*omero.sessions* property), [273](#)
[sync_on_login](#) (*omero.ldap* property), [268](#)
[system_memory](#) (*omero.jvmcfg* property), [266](#)

T

[tcp](#) (*omero.ports* property), [281](#)
[template_dirs](#) (*omero.web* property), [303](#)
[threads](#) (*omero.pixeldata* property), [279](#)
[thumb_default_size](#) (*omero.client.browser* property), [255](#)
[thumbnails_batch](#) (*omero.web* property), [303](#)
[tile_height](#) (*omero.pixeldata* property), [279](#)
[tile_sizes_bean](#) (*omero.pixeldata* property), [279](#)
[tile_width](#) (*omero.pixeldata* property), [280](#)
[time_zone](#) (*omero.web* property), [303](#)
[timeout](#) (*omero.mail.smtp* property), [271](#)
[timeout](#) (*omero.query* property), [282](#)
[timeout](#) (*omero.scripts* property), [284](#)
[timeout](#) (*omero.sessions* property), [274](#)
[timeout](#) (*omero.web.public.cache* property), [297](#)
[top_links](#) (*omero.web.ui* property), [304](#)
[top_logo](#) (*omero.web* property), [303](#)
[top_logo_link](#) (*omero.web* property), [304](#)
[trustStore](#) (*omero.security* property), [289](#)
[trustStorePassword](#) (*omero.security* property), [289](#)
[twitter](#) (*omero.web.sharing* property), [302](#)
[type_order](#) (*omero.client.ui.tree* property), [258](#)

U

[update\(\)](#) (*omero.grid.Table* method), [479](#)
 Upload Scripts, [321](#)
[url](#) (*omero.db* property), [261](#)
[url_filter](#) (*omero.web.public* property), [298](#)
[urls](#) (*omero.ldap* property), [269](#)
[use_x_forwarded_host](#) (*omero.web* property), [305](#)
[user](#) (*omero.db* property), [262](#)
[user](#) (*omero.web.public* property), [298](#)
[user_dropdown](#) (*omero.web* property), [305](#)
[user_filter](#) (*omero.ldap* property), [269](#)
[user_mapping](#) (*omero.ldap* property), [269](#)
[username](#) (*omero.ldap* property), [269](#)
[username](#) (*omero.mail* property), [272](#)

V

[values](#) (*omero.grid.DoubleColumn* attribute), [477](#)
 VERBOSE, [444](#)
[VerifyPeer](#) (*omero.glacier2.IceSSL* property), [263](#)
[version](#) (*omero.db* property), [262](#)
 View, [314](#)

`view` (*omero.web.viewer property*), 305

W

`warn` (*omero.throttling.method_time property*), 275

`webgateway_cache` (*omero.web property*), 305

Write Data, 321

`ws` (*omero.ports property*), 281

`wsgi_args` (*omero.web property*), 305

`wsgi_timeout` (*omero.web property*), 306

`wsgi_workers` (*omero.web property*), 306

`wss` (*omero.ports property*), 281

X

`x_frame_options` (*omero.web property*), 306