OMERO Documentation

Release 4.4.12

The Open Microscopy Environment

September 23, 2014

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

The OMERO 4.4.12 documentation is divided into three parts. *About the OMERO Platform* introduces the user-facing client applications and how to get started, as well as detailing where users can access further help and support. System administrators wanting to install an OMERO server can find instructions in the *System Administrator Documentation*. Finally, developers can find more specific and technical information about OMERO in the *Developer Documentation*.

Additional online resources can be found at:

- Downloads[1]
- Features (includes movie tutorials)[2]
- Screenshots[3]
- Security Vulnerabilities[4]
- ***NEW*** User help website[5]
- ***NEW*** Script sharing service[6]
- ***NEW*** Partner projects to extend OMERO[7]

OMERO version *4* uses the *June 2012* release of the OME-Model[8].

Note: With the release of OMERO 5.0, the 4.4.x line has now entered maintenance mode. We will continue to support this version throughout 2014 but it will only be updated for major bug fixes.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

[1] http://downloads.openmicroscopy.org/latest/omero4/
[2] http://www.openmicroscopy.org/site/products/omero/feature-list
[3] http://www.openmicroscopy.org/site/products/omero/screenshots
[4] http://www.openmicroscopy.org/site/products/omero/secvuln
[5] http://help.openmicroscopy.org/
[6] http://www.openmicroscopy.org/site/community/scripts
[7] http://www.openmicroscopy.org/site/products/partner/
[8] http://www.openmicroscopy.org/site/support/ome-model/

Part I

About the OMERO Platform

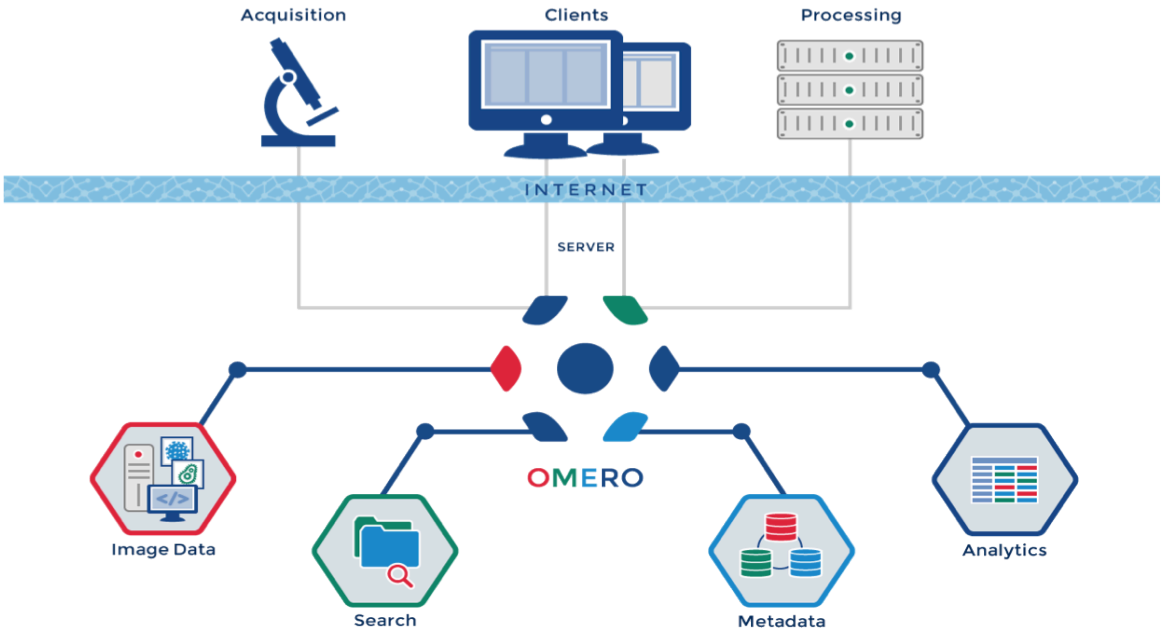
CHAPTER

ONE

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 1.6 or higher to be installed. This can easily be installed from http://java.com/en 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 latest version of OMERO 4.4.x. We also have archived versions available for previous versions of OMERO[1]. For more technical information, please refer to the *Developer Documentation*.

Note: With the release of OMERO 5.0, the 4.4.x line has now entered maintenance mode. We will continue to support this version throughout 2014 but it will only be updated for major bug fixes..

[1] http://www.openmicroscopy.org/site/support/previous/

CHAPTER

TWO

RESOURCES

- There are a number of demonstration videos available on the Features List[1] page, providing an overview of the applications.
- 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.
- ***NEW*** Our partners within the OME consortium are working on integrating additional functions and modules with OMERO. See the Partner Projects[2] page for details of the latest extensions which could help OMERO meet your research needs more fully.
- ***NEW*** You can also extend the functionality of OMERO using OMERO.scripts, our version of plugins. Guides to some of the scripts which ship with OMERO releases are already provided, but you can also check out our Script Sharing[3] page to find extra ones.
- ***NEW*** Workflow-based user assistance guides are provided on our help website[4]. Extended OMERO.web and guides to other OMERO applications will be coming here soon.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

2.1 Community support

The Open Microscopy Environment[5] 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.

2.1.1 Web

The Open Microscopy Environment website is at http://www.openmicroscopy.org/site. Bio-Formats can be found at http://www.openmicroscopy.org/site/products/bio-formats.

2.1.2 Mailing lists

The following lists are provided:

- ome-users[6] – support with installation and general use or miscellaneous queries, as well as bug reporting
- ome-devel[7] – development discussion and support

Note: Both of these lists are moderated and only allow posts from subscribers. Please subscribe to the lists to participate in the discussion.

[1] http://www.openmicroscopy.org/site/products/omero/feature-list
[2] http://www.openmicroscopy.org/site/products/partner/
[3] http://www.openmicroscopy.org/site/community/scripts
[4] http://help.openmicroscopy.org/
[5] http://www.openmicroscopy.org/site
[6] http://lists.openmicroscopy.org.uk/mailman/listinfo/ome-users/
[7] http://lists.openmicroscopy.org.uk/mailman/listinfo/ome-devel/

2.1.3 Forums

Discussion on a number of topics is also available through our forums[8]. Forums include:

- OME Announcements[9]
- OMERO[10]
  - User Discussion[11]
  - Installation and Deployment[12]
  - Developer Discussion[13]
- Bio-Formats[14]
  - User Discussion[15]
- OME Data Model[16]
  - User Discussion and Suggestions[17]

[8] http://www.openmicroscopy.org/community/
[9] http://www.openmicroscopy.org/community/viewforum.php?f=11
[10] http://www.openmicroscopy.org/community/viewforum.php?f=3
[11] http://www.openmicroscopy.org/community/viewforum.php?f=4
[12] http://www.openmicroscopy.org/community/viewforum.php?f=5
[13] http://www.openmicroscopy.org/community/viewforum.php?f=6
[14] http://www.openmicroscopy.org/community/viewforum.php?f=12
[15] http://www.openmicroscopy.org/community/viewforum.php?f=13
[16] http://www.openmicroscopy.org/community/viewforum.php?f=14
[17] http://www.openmicroscopy.org/community/viewforum.php?f=15

CHAPTER

THREE

OMERO CLIENTS

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

3.1 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**, **OMERO.editor** 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 (.editor, .importer) are desktop applications written in Java and require Java 1.6 (or higher) to be installed on the user's computer (automatically installed on most up-to-date OS X and Windows XP systems).

Our **NEW** user assistance help website[1] provides a series of workflow-based guides to performing common actions in the client applications, such as importing and viewing data, exporting images and using the measuring tool.

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 Partner Projects[2] page for more details.

3.1.1 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 [3] [4]. Developers can use the *OMERO.web framework* to build customized views.

For more information and guides to using OMERO.web, see our help website[5].

3.1.2 OMERO.insight

OMERO.insight provides a number of tools for accessing and using data in an OMERO server. The figure *OMERO.insight ImageViewer* presents the OMERO.insight image viewer, whereas figure *OMERO.insight* presents the user interface [6] [7]. To find out more, see the OMERO.insight user guides[8].

Among many features, the noteworthy OMERO.insight elements are:

- DataManager, a traditional tree-based view of the data hierarchies in an OMERO server. DataManager supports access to all image metadata, annotations, tags.

[1] http://help.openmicroscopy.org/
[2] http://www.openmicroscopy.org/site/products/partner/
[3] Krenn, et al., JCB (http://jcb.rupress.org/content/196/4/451)
[4] Snider, et al., JCB (http://jcb.rupress.org/content/195/2/217)
[5] http://help.openmicroscopy.org/
[6] Dantas, et al., JCB (http://jcb.rupress.org/content/193/2/307)
[7] Roscioli, et al., JCB (http://jcb.rupress.org/content/196/4/435)
[8] http://help.openmicroscopy.org/

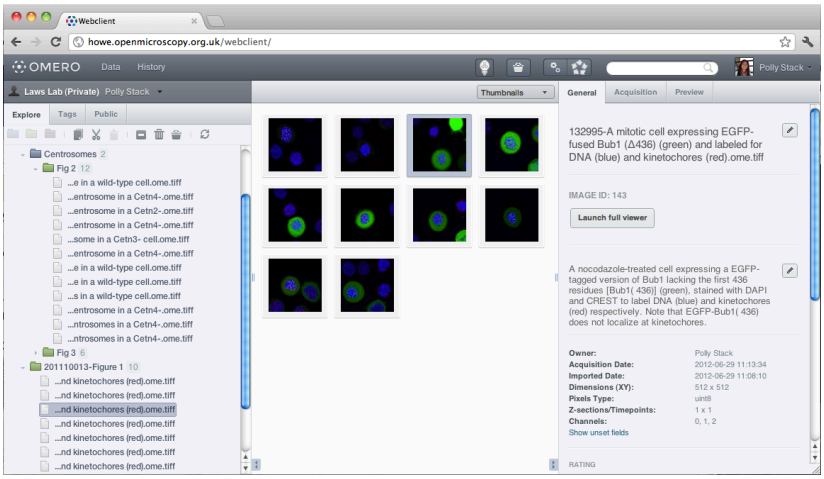


Figure 3.1: OMERO.web user interface

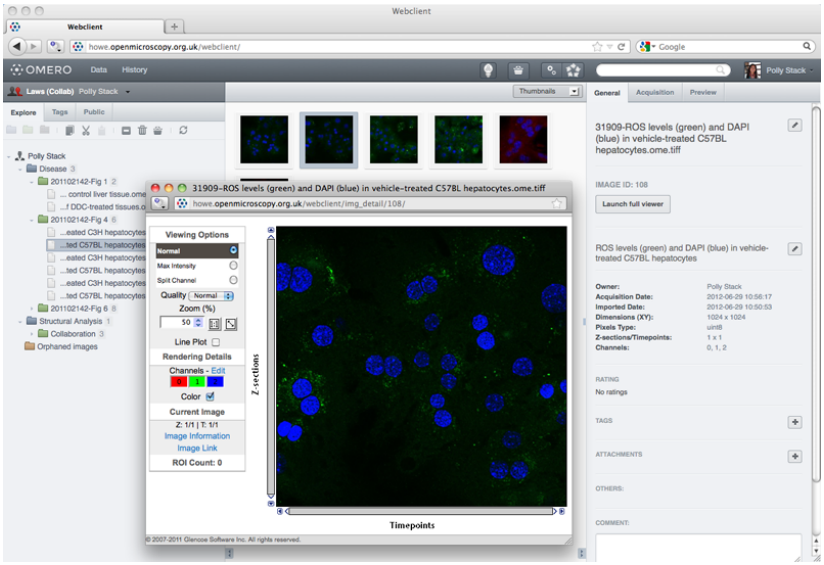


Figure 3.2: OMERO.web image viewer

- 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

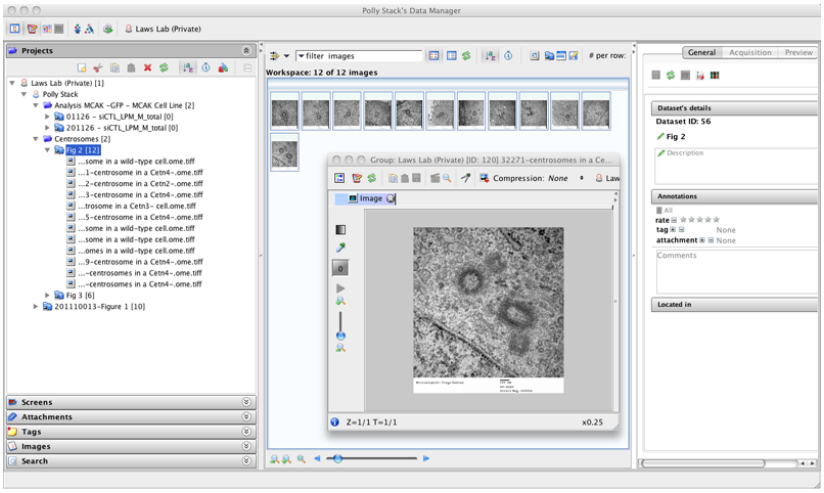


Figure 3.3: OMERO.insight ImageViewer

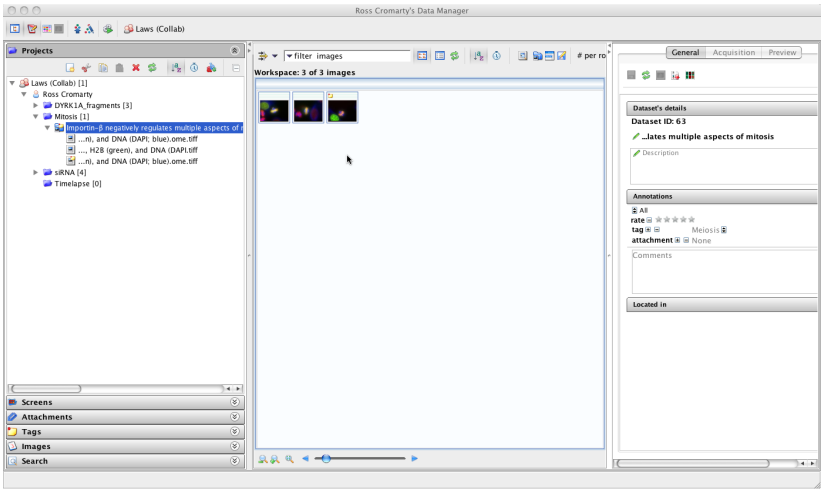


Figure 3.4: OMERO.insight

viewing at remote locations is enabled. Image rendering settings are saved and chosen by user ID.

- Measurement Tool, a sub-application of ImageViewer that enables size and intensity measurements of defined regions-of-interest (ROIs).
- Working Area, for viewing, annotating, and manipulating large sets of image data.
- User administration.
- Image import.

Our user assistance help website[9] features a number of workflow-based guides to importing, viewing, managing and exporting your data using OMERO.insight.

3.1.3 OMERO.editor

OMERO.editor is an editing tool designed to facilitate recording of experimental metadata, for annotation of images in OMERO, where users can create a "template" (for example, to describe a protocol) and then use this template to create individual "experiment" files, which contain the experimental metadata. A summary of the experiment can be viewed alongside annotated images in OMERO.insight. This workflow makes it easy to reuse protocols, and to build up a detailed description of an experiment by combining several smaller protocols.

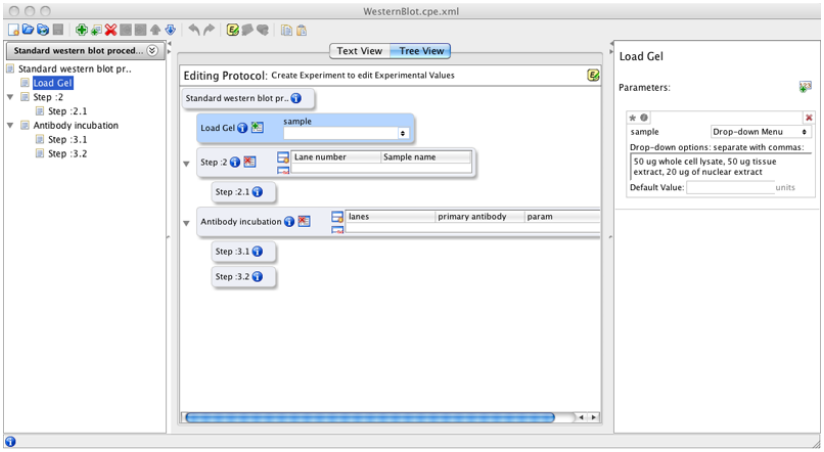


Figure 3.5: OMERO.editor

[9] http://help.openmicroscopy.org/

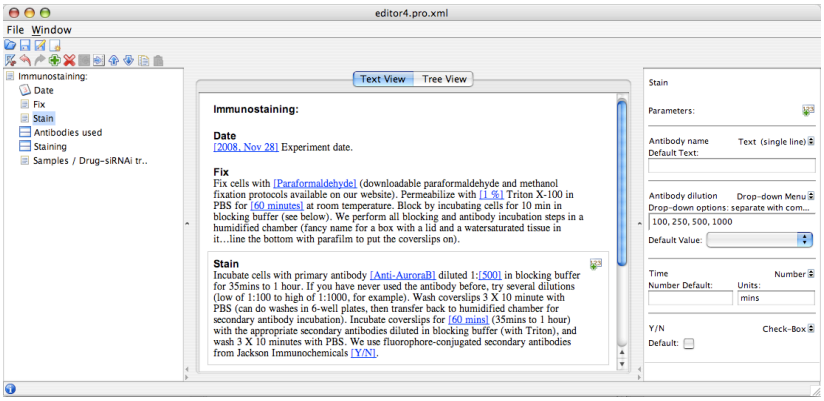


Figure 3.6: OMERO.editor

The OMERO.editor is part of the OMERO.insight client, but can also run as a stand-alone application. OMERO.editor saves files as XML documents, which makes it possible for them to be read by other software. More information can be found in the OMERO.editor user guide[10].

Note: OMERO.editor is no longer actively developed but is still shipped with OMERO releases.

3.1.4 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[11] for a detailed list of supported formats.

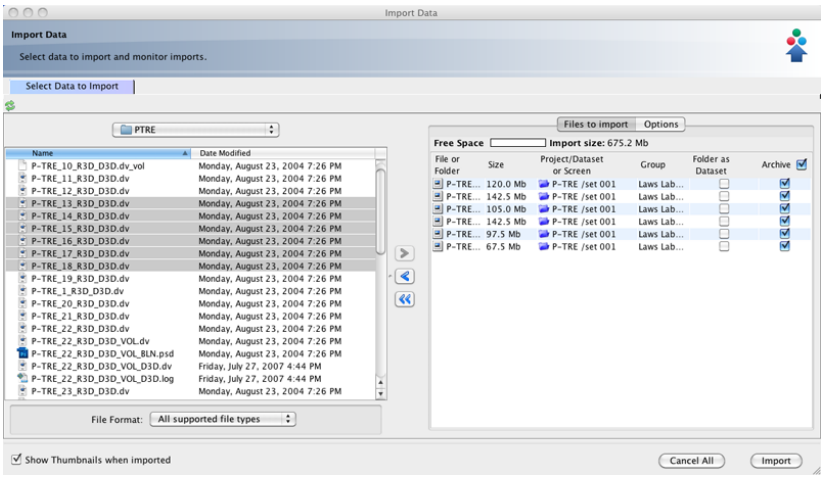


Figure 3.7: OMERO.importer

Citation

The screenshots make use of data from the JCB DataViewer[12] under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License. For more information see Attribution[13].

[10] http://help.openmicroscopy.org/editor.html
[11] http://www.openmicroscopy.org/site/support/bio-formats4/supported-formats.html
[12] http://jcb-dataviewer.rupress.org/
[13] http://www.openmicroscopy.org/site/about/licensing-attribution/attribution

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

3.2 OMERO Command Line Interface

See also:

OMERO Command Line Interface System administrator documentation for the Command Line Interface

OMERO Command Line Interface Developer documentation for the Command Line Interface

3.2.1 Overview

The CLI (Command Line Interface) is a set of Python[14] 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.

Requirements

Check you have Python[15] installed by typing:

```
$ python --version
Python 2.5.1
```

Additionally, Ice[16] must be installed on your machine:

```
$ python -c "import Ice"
```

The CLI is currently bundled with the OMERO.server. Download the version corresponding to your system from the OMERO downloads[17] page.

Note: The CLI is bundled with the OMERO.server but that does not imply you must use that directory as a server. You can download the server zip to a number of machines and use the CLI commands from each machine to access an existing OMERO instance.

Once the server is downloaded, the CLI is located under the `bin/` directory:

```
$ cd OMERO.server
$ bin/omero -h
OMERO Python Shell. Version 4.4.5-ice33
Type "help" for more information, "quit" or Ctrl-D to exit
omero>
```

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:

```
$ bin/omero -h
```

[14] http://python.org
[15] http://python.org
[16] http://www.zeroc.com
[17] http://downloads.openmicroscopy.org/latest/omero4/

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

```
$ bin/omero admin -h
$ bin/omero admin start -h
```

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

```
$ bin/omero help debug       # debug is an option
$ bin/omero help admin       # same as bin/omero admin -h
```

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. The connection parameters can be either passed directly in the connection string:

   ```
   $ bin/omero login username@servername:4064
   ```

   or using the *-s*, *-u* and *-p* options:

   ```
   $ bin/omero login -s servername -u username -p 4064
   ```

   If no argument can be specified, the interface will ask for the connection credentials:

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

   ```
   $ bin/omero import image.tiff
   ```

2. By passing the session arguments directly to the desired command, e.g.:

   ```
   $ bin/omero -s servername -u username -p 4064 import image.tiff
   ```

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

   ```
   $ bin/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:

```
$ bin/omero logout
```

3.2.2 Import images

`omero import` is probably the first command many users will want to use. To import a file image.tiff, use:

```
$ bin/omero import image.tiff
```

Many options can be passed to the `omero import`. They can be listed using the *-h* option:

```
$ bin/omero import -h
```

3.2.3 Manage sessions

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

```
$ bin/omero sessions -h
```

Multiple sessions

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

```
$ bin/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:
 localhost | root | system          | 1f800a16-1dc2-407a-8a85-fb44005306be | True      | Fri Nov 23 14:
(2 rows)
```

Sessions keys can then be reused to switch between stored sessions:

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

Sessions directory

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

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

If you want to use a custom session directory, use the *--session-dir* argument in the `omero sessions` commands:

```
$ bin/omero login --session-dir=/tmp
$ bin/omero sessions list --session-dir=/tmp
$ bin/omero logout --session-dir=/tmp
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

3.3 The Command Line Import

The Command Line Importer tool (CLI) 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 files formats and functions in much the same way. Visit Supported Formats[18] for a detailed list of supported formats.

3.3.1 Starting the Command Line Importer

There are two ways to use the importer:

- Either by using the included shell scripts for Linux and Macintosh,
- or by calling directly the `ome.formats.importer.cli.CommandLineImporter` class from java on the command line (you will also need to include a path to the required support jars - look inside of the importer-cli scripts for an example of how to do this.)

An example of starting the importer from the command line might look like this:

```
./importer-cli …
```

3.3.2 From server installation

```
#!sh
bin/omero -s localhost -u user import …
```

This can also be used to detect what a command ‘’would” import.

```
#!sh
bin/omero import -f /path/to/file
```

will print on standard out a list of all the files which would be imported in groups separated by “#” comments.

3.3.3 Command Line options

The Command Line Importer tool takes a number of mandatory and optional arguments to run, as follows:

```
Usage:  importer-cli [OPTION]... [DIR|FILE]...
or:   importer-cli [OPTION]... -

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
```

[18] http://www.openmicroscopy.org/site/support/bio-formats4/supported-formats.html

```
all valid imports.

Mandatory arguments:
 -s   OMERO server hostname
 -u   OMERO experimenter name (username)
 -w   OMERO experimenter password
 -k   OMERO session key (can be used in place of -u and -w)
 -f   Display the used files (does not require other mandatory arguments)

Optional arguments:
 -c   Continue importing after errors
 -a   Archive the original file on the server
 -l   Use the list of readers rather than the default
 -d   OMERO dataset Id to import image into
 -r   OMERO screen Id to import plate into
 -n   Image name to use
 -x   Image description to use
 -p   OMERO server port [defaults to 4064]
 -h   Display this help and exit

 --no_thumbnails                                    Do not perform thumbnailing after import
 --plate_name                                       Plate name to use
 --plate_description                                Plate description to use
 --debug[=ALL|DEBUG|ERROR|FATAL|INFO|TRACE|WARN]    Turn debug logging on (optional level)
 --report                                           Report errors to the OME team
 --upload                                           Upload broken files with report
 --logs                                             Upload log file with report
 --email                                            Email for reported errors
 --annotation_ns                                    Namespace to use for subsequent annotation
 --annotation_text                                  Content for a text annotation (requires namespace)
 --annotation_link                                  Comment annotation ID to link all images to

 e.g. importer-cli -s localhost -u bart -w simpson -d 50 foo.tiff

 Report bugs to <ome-users@lists.openmicroscopy.org.uk>
```

These options will also be displayed on the command line by passing no arguments or "-h" to the importer.

Note:

- Using the `--report` option sends an automated error report to the QA application. HTTP POST requests are currently used to upload the report. The default parameters (eg. endpoint URL) may be overridden via an INI-formatted configuration file, which is expected to be located within a `config` directory relative to the CLI importer (see example below).
- The `--email` option is the OMERO user's contact email. Note that errors are *not* sent to this address.

Sample `config/importer.config` INI file:

```
[General]
appTitle = OMERO.importer
appVersionNote =
port = 4064
disableUpgradeCheck = false

[Uploader]
TokenURL = http://qa.openmicroscopy.org.uk/qa/initial/
URL = http://qa.openmicroscopy.org.uk/qa/upload_processing/
BugTrackerURL = http://qa.openmicroscopy.org.uk/qa/upload_processing/
forumURL = http://www.openmicroscopy.org/community/

[UI]
forceFileArchiveOn = false
disableImportHistory = false
```

CHAPTER

FOUR

QUICKSTART SERVER ACCESS

If your institution does not have an existing OMERO.server for you to connect to, you can create your own using a virtual appliance (a step-by-step guide for how to do this is provided). We also have a demo service but this has been updated to our new OMERO 5 line as of the end of January 2014.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

4.1 OMERO virtual appliance

The OMERO virtual appliance is a quick, easy, and low-cost way to try out OMERO.server on your laptop or desktop. This enables you to make an informed decision about whether committing to an OMERO.server install is right for you.

Virtualization enables canned, ready to run software environments to be created and used, in the form of VM (Virtual Machine), or to be distributed for others to use, in the form of virtual appliances. A Virtual Appliance is essentially a file that describes how to create a new Virtual Machine on demand. The virtualized software environment can contain an entire OS (Operating System), such as Windows or Linux, and any other software that runs in that OS, such as, in this case, OMERO.server and its associated software prerequisites. Once created and started, you can log into the OS and use it as though it were a real machine. One way to think of this is as though you had an entire computer in a window on your desktop.

When using virtualization software, the OS that is running the virtualization software is referred to as the "**host OS**". When you use virtualization, the OS running within a virtual machine is referred to as the "**guest OS**". This allows us to be explicit about which OS we are working in.

This technology allows the OME Project to distribute a canned, ready-to-run environment containing an OMERO.server, freeing you from having to install the server and prerequisites yourself, and letting you concentrate on evaluating the functionality of the OMERO platform.

Note: The virtual hard-drive used by the OMERO virtual appliance is 30GB in size and you should keep track of the amount of this space you have consumed and, if necessary, delete data that is not required. If your data is likely to exceed this space whilst you are evaluating OMERO then it is worthwhile going through the *Increasing HD size* before you start working with OMERO in earnest.

4.1.1 Getting started

To use the virtual appliance you should do the following:

- Install VirtualBox
- Download the OMERO.server virtual appliance
- Import the virtual appliance into VirtualBox to create a virtual machine
- Start the virtual machine

Each of these points is outlined in more detail below.

Install VirtualBox

Download VirtualBox from the VirtualBox Downloads page[1] and follow the installation process for your platform. If in doubt, you should download, or upgrade to, the latest version of VirtualBox. Once VirtualBox is installed, run the application. Depending upon your platform and version, the VirtualBox interface should look similar to the following screenshot:

Figure 4.1: VirtualBox installation

Download the OMERO.server virtual appliance

The virtual appliance can be downloaded from the OMERO download page[2] and should have a filename similar to, e.g. omero-vm-4.4.12.ova

Import OMERO virtual appliance into VirtualBox

- Start VirtualBox then select **'File/Import Appliance'**. You will be presented with a dialogue box.
- Select and navigate to the location where you downloaded the the virtual appliance file.
- Select your OVA file then click **open**.

This process is indicated in the screenshot below.

- Click **continue**. You will be presented with a range of options for the VM that will be built from the appliance.
- You can accept the defaults by clicking **Import**.

You should now see a progress bar as your new virtual machine is built from the appliance. This may take a few minutes depending upon your hardware.

When the import procedure is complete, your new VM should appear in the VirtualBox VM library ready for use.

Networking

Our virtual appliance is distributed with VirtualBox's built in Host-Only Network Address Translation (NAT) preconfigured. This means that the IP address for the VM is 10.0.2.15 as this is the default VirtualBox Host-Only NAT address. Using this address is the simplest way to distribute a virtual appliance when you do not know the setup of a user's network.

[1] https://www.virtualbox.org/wiki/Downloads
[2] http://downloads.openmicroscopy.org/latest/omero4/

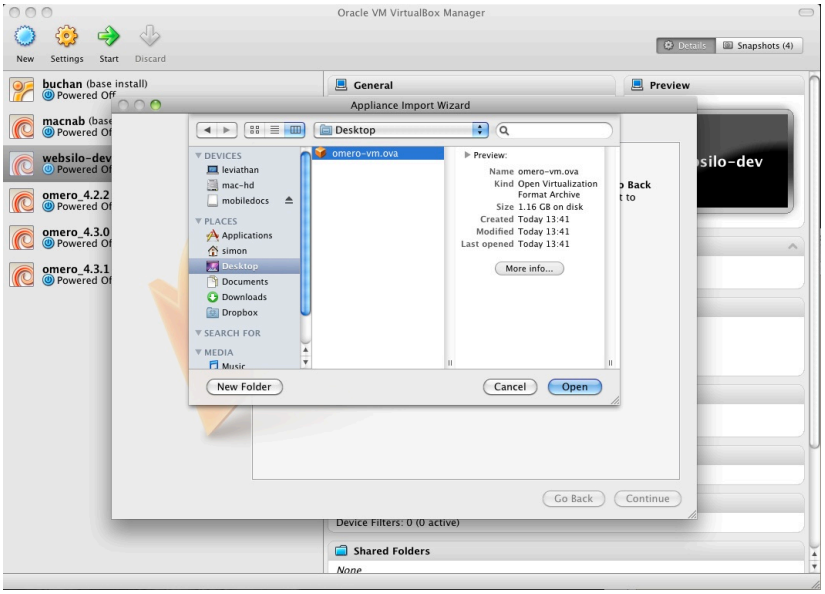


Figure 4.2: Import of the OMERO virtual appliance

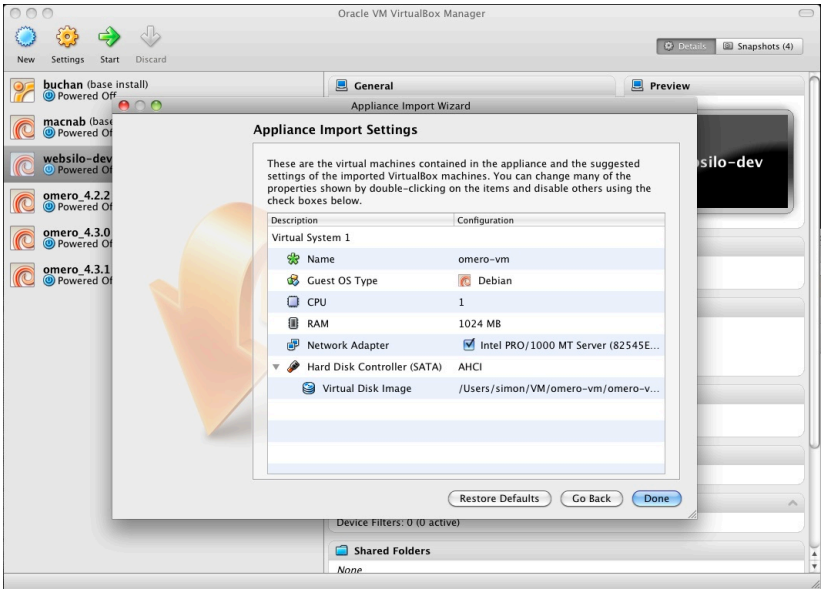


Figure 4.3: Virtual appliance import settings

Port-forwarding settings

Your host OS cannot connect directly to 10.0.2.15 but needs to use port-forwarding. This means that you connect to your localhost on a specific port and the communications to and from that port are forwarded to specified ports on the guest VM.

Our virtual appliance should be preconfigured with the correct port-forwarding setting during the import process. However, it is best to double check that these settings are correct:

- Select your VM in the VirtualBox VM Library
- Click on **Settings** then select the **Network** tab
- Click on **Advanced**
- Click on **Port Forwarding**

If the table in the window that appears is empty then port forwarding is not setup. The required port-forwarding settings are as follows:

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
omero-ssl	TCP	127.0.0.1	4064	10.0.2.15	4064
omero-unsec	TCP	127.0.0.1	4063	10.0.2.15	4063
omero-web	TCP	127.0.0.1	8080	10.0.2.15	8080
ssh	TCP	127.0.0.1	2222	10.0.2.15	22

When correctly setup in VirtualBox, your port forwarding settings should look like this:

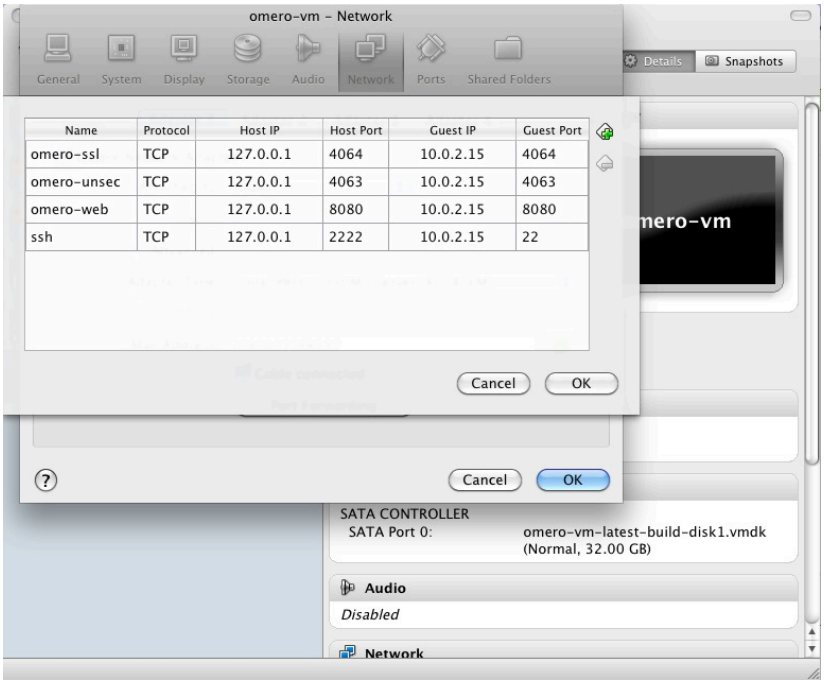


Figure 4.4: VirtualBox port forwarding

If you are on Linux or Mac OS X, you can either use our port forwarding setup script or you can set up port forwarding manually.

On Microsoft Windows systems you will have to set up port forwarding manually as the script requires a Bash shell.

The script can be downloaded from the online version of this documentation; see http://www.openmicroscopy.org/site/support/omero/users/virtual-appliance.html.

After obtaining the script, it can be used in the following manner:

```
$ bash setup_port_forwarding.sh $VMNAME
```

where $VMNAME is the name of your VM.

Note: By default the scripts create a VM named **omerovm** and the pre-built appliance is named **omero-vm**

Adding port forwarding manually is achieved by editing the port forwarding table shown above. Use the + to add a new row to the table, then click in each cell and type in the required settings.

Now you are ready to start your VM. Select the VM in the VirtualBox VM library then click **start**.

A window should open containing a console for your VM which should now be going through its standard boot process. OMERO.server is automatically started at boot time, meaning that you should be able to interact with OMERO without further setup.

Credentials

There are a number of accounts that are preconfigured in the OMERO virtual appliance. Two of these are OS accounts, for logging into the VM as either the **root** user or the **omero** user. There is also a single OMERO.server account which is used to access the OMERO.server software as the OMERO.server **root** user.

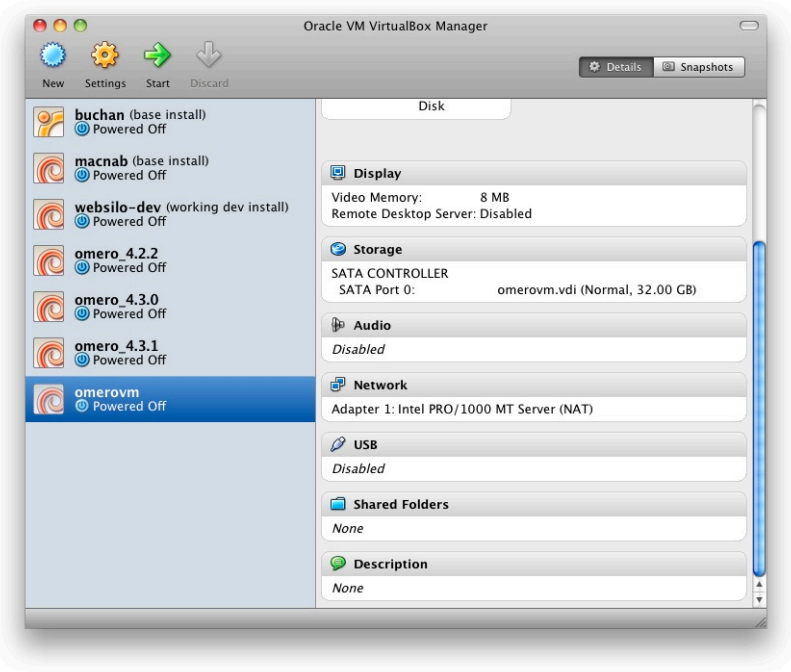


Figure 4.5: VirtualBox VM manager

Figure 4.6: Booting the virtual appliance

Virtual Appliance OS credentials

Username	Password
root	[1]
omero	omero

[1] The omero account is able to run administrative commands via sudo with no password required.

OMERO.server credentials

Username	Password
root	omero

You can use this administrative account to create as many user level accounts as you require in the usual way.

4.1.2 Working with the OMERO.VM

Now that your VM is up and running you have a choice about how to interact with it.

- You can connect to OMERO.web from your host browser. Go to http://localhost:8080/webclient.
- You can **use OMERO.clients from within your host OS.** This will allow you to import data via a GUI and manage that data once imported. To do so, download the OMERO.insight client[3] and follow the instructions below. More information can be found on our help website[4] which provides workflow-based guides to using the OMERO.clients.
- Alternatively, you can interact with the server command line interface by SSH (Secure Shell)'ing into the guest VM or by opening a console within the VM itself. Administrators may need to use one of these methods to restart the server and/or change configuration parameters. In this case, you must have an SSH client installed on your host machine to use to connect to the OMERO.server.

Note: The following examples assume that the OMERO VM is up and running on the same machine that you are working on.

OMERO.web

Go directly to http://localhost:8080/webclient to log in with user: "root" / pw: "omero".

Note: If you receive a 502 nginx error on first attempting to connect to the web app on http://localhost:8080/webclient/ please restart the virtual machine and try again.

OMERO.insight

You can run regular OMERO clients on your host machine and connect to the server in the VM. Our example uses OMERO.insight running on Mac OS X to connect to the VM.

- Download[5] and install OMERO.insight
- Start OMERO.insight
- Click the spanner icon situated above the password box to enter the server settings box shown below.
- Use the + icon to add a new server entry with the address *localhost* and the port *4064* then click apply
- You can now use the login credentials given above to log into OMERO.insight using the login window shown below (user: "root" / pw: "omero").
- OMERO.insight should now load up and display the main window.

You can now use OMERO.insight to import and manage images on a locally running virtual server just like you would use the standard remote server.

Note: A Getting Started guide is available for OMERO.insight on our help website[6] if you need further assistance to download and install the software.

---

[3] http://downloads.openmicroscopy.org/latest/omero4/
[4] http://help.openmicroscopy.org
[5] http://downloads.openmicroscopy.org/latest/omero4/
[6] http://help.openmicroscopy.org/getting-started-4.html

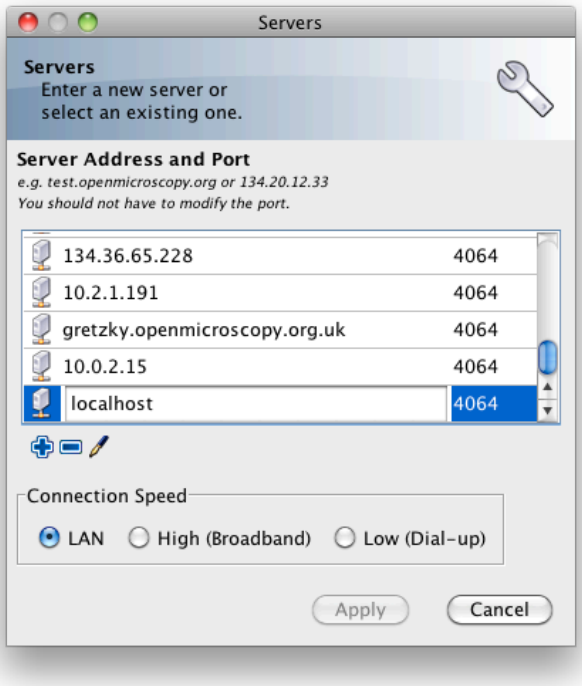


Figure 4.7: Setting OMERO.insight server address and port number

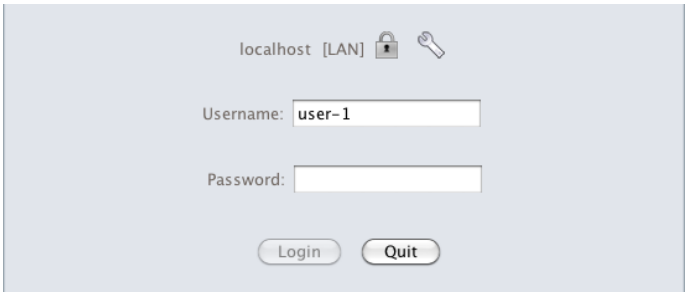


Figure 4.8: OMERO.insight login window

Secure Shell

You can log into your VM using SSH, allowing you to use the *OMERO Command Line Interface*. In the following example, we assume that you have an SSH client installed on your host machine and also that your VM is up and running.

You can log into the VM using the above credentials and the following command typed into a terminal:

```
$ ssh omero@localhost -p 2222
```

This invokes the SSH program telling it to login to the localhost on port 2222 using the username *omero*. Remember that earlier you set up port forwarding to forward port 2222 on the host machine to port 22 (the default SSH port) on the guest VM. You should be prompted for a password. Once you have successfully entered your password, you should be greeted by a prompt similar to the following:

```
omero@omerovm:~$
```

There are two potential complications to this method, (1) if you have used a VM before then there could be old SSH fingerprints set up, (2) the first time that you log into the VM you will be asked to confirm that you wish to continue connecting. If you get the following message after you invoke SSH, you need to remove the old fingerprints:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
60:e0:d2:e8:fb:25:bf:09:53:9d:9d:59:59:45:cf:aa.
Please contact your system administrator.
Add correct host key in /Users/rleigh/.ssh/known_hosts to get rid of this message.
Offending key in /Users/rleigh/.ssh/known_hosts:14
RSA host key for localhost has changed and you have requested strict checking.
Host key verification failed.
```

You can do this using the following command typed into the terminal:

```
$ ssh-keygen -R [localhost]:2222 -f ~/.ssh/known_hosts
```

This should produce output similar to:

```
$ ssh-keygen -R [localhost]:2222 -f ~/.ssh/known_hosts
/Users/rleigh/.ssh/known_hosts updated.
Original contents retained as /Users/rleigh/.ssh/known_hosts.old
```

The first time that you log into the VM you will also be asked to confirm that you wish to connect to this machine by a message similar to the following:

```
$ ssh omero@localhost -p 2222
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be established.
RSA key fingerprint is 60:e0:d2:e8:fb:25:bf:09:53:9d:9d:59:59:45:cf:aa.
Are you sure you want to continue connecting (yes/no)?
```

You should confirm that you wish to continue connecting, after which you will be prompted for your password as usual:

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2222' (RSA) to the list of known hosts.
omero@localhost's password:
```

After which, you should have a prompt indicating that you have a shell open and logged into the VM:

```
omero@localhost's password:
Linux omerovm 2.6.32-5-686 #1 SMP Mon Jun 11 17:24:18 UTC 2012 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Apr  5 10:32:18 2012 from 10.0.2.2
omero@omerovm:~$ _
```

Log into the VM directly

Note: Due to the frequent changes in the VirtualBox Guest Additions, key mappings between the host and guest OS do not

always work. We recommend using SSH as the primary way of interacting with the Virtual Appliance CLI.

When you start your VM using the VirtualBox GUI, as outlined above, a window will be displayed showing the boot process for the machine as it starts up, just like with a real piece of hardware. Once the boot process has finished you will see a prompt displayed in this window like so:

```
[System startup messages]

Debian GNU/Linux 6.0 omerovm tty1

omerovm login: _
```

You can log into the console of the VM directly using the user account credentials above.

```
omerovm login: omero
Password: _
```

There is no GUI on the current OMERO virtual appliance so you will have to use the Bash shell which looks like this:

```
omero@omerovm:~$ _
```

From here you can interact with OMERO.server via the *OMERO Command Line Interface*. You will need to login as the 'omero' user to access the OMERO CLI (user: "omero" / pw: "omero"). Logout using Ctrl-D.

4.1.3 Known issues

Networking not working

Occasionally, during the boot process, the VirtualBox DHCP server fails to allocate an IP address to the OS in the guest VM. This means that OMERO.clients, such as OMERO.insight, cannot connect to the OMERO.server in the VM.

- **CAUSE:** We believe that this is an intermittent VirtualBox bug that resurfaces across many versions VirtualBox #4038[7] and previously VirtualBox #3655[8]
- **DIAGNOSIS:** Check whether the guest VM has been allocated the reserved host-only NAT IP address. If 10.0.2.15 does not appear in the output from **ifconfig** then this issue has occurred. The easiest way to verify this is to log into the guest VM console and check the output from executing the following command:

  ```
  $ ifconfig
  ```

- **SOLUTION:** An easy, but unreliable, fix is to reboot the guest VM. The preferred fix is to log into the guest VM console and execute the following commands, which will cause the guest OS to release its IP lease before requesting a new lease:

  ```
  $ dhclient -r
  $ dhclient -eth0
  ```

Port conflict when OMERO.server already running in Host OS

If you are already running an instance of the OMERO.server in your host OS then there will be a conflict due to the ports assigned to VirtualBox port-forwarding already being in use.

- **SOLUTION 1:** Turn off the OMERO.server in the host environment by issuing the following command:

[7] https://www.virtualbox.org/ticket/4038
[8] https://www.virtualbox.org/ticket/3655

```
$ omero admin stop
```

- **SOLUTION 2:** Alter the port-forwarding settings for your OMERO.VM as described in the *Port-forwarding settings* section. For example, increment the host port settings for omero-ssl, omero-unsec, and omero-web.

Note: We are assuming that your host OS is not already running services on those ports. You can check whether something is already listening on any of these ports by running the following commands (Mac OS X) which should return the prompt without any further output if there is nothing listening:

```
$ lsof -nP | grep -E '(:4063)|(:4064)'
```

VM will not boot because the HDD (Hard Disk Drive) is full

If you fill the virtual HDD used by your VM then the OS may be unable to boot and you will lose access to your OMERO.server install. You may also get a "error 28: no space left on device" message. To log into your VM you will need to use the recovery mode. Start the VM and at the Grub screen, use the down arrow followed by return to select the recovery mode entry, e.g.

```
Debian GNU/Linux, with Linux 2.6.32-5-686 (recovery mode)
```

as illustrated in this example of the Grub screen:

```
                GNU GRUB  version 1.98+20100804-14
 ------------------------------------------------------------------------
|Debian GNU/Linux, with Linux 2.6.32-5-686                               |
|Debian GNU/Linux, with Linux 2.6.32-5-686 (recovery mode)               |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
 ------------------------------------------------------------------------
```

Do not worry if your VM has a kernel number different to 2.6.32-5-686, the important thing is that you select the entry labeled "recovery mode". At this point, the VM should rapidly boot into the recovery mode and enable you to log in using the root password.

Once you have logged in, you have a number of options but the recommended courses of action are:

1. Delete unnecessary files using standard Linux command line tools like *rm* to make space for the VM to boot normally, then use your favored OMERO client to login and delete more files. A useful place to start might be by deleting the logs stored in /var/logs.
2. Increase the size of your virtual HDD. If you have filled your existing HDD then it is likely that the volume of data you are storing is too big for the default HDD. You should follow the instructions in the *Increasing HD size* section to ensure that the size of virtual HDD you have available is suitable for the volumes of data that you are collecting.

4.1.4 Increasing HD size

Image data can become very large and easily fill available hard-drive space. By default, the OMERO virtual appliance is supplied with a 30GB virtual hard-drive. Before using the appliance, consider the volume of data you will be working with whilst evaluating OMERO and whether you need to increase the size of the virtual hard-drive to accommodate it.

The following is a step-by-step guide; be aware that this is not a risk-free procedure and you should backup your VM before proceeding.

Preliminary steps

Acquiring a Ubuntu Linux ISO

Download an Ubuntu Linux ISO[9]. The most up-to-date version is fine.

Backing up your VM

Before you proceed further, you should create a clone of the omero-vm and subsequently work on the copy so that if something gets broken you can always start again. The easiest way to do this is from the command line.

Note: If you are on Windows then you should navigate to `C:\Program Files\Oracle\VirtualBox\` because the VBoxManage tools are not added to your path by default.

Start a shell and, assuming that your VM has the default name of omero-vm, use the following command:

```
$ VBoxManage clonevm omero-vm --mode machine --options keepallmacs --name omero-vm-2 --register
```

This will create a copy of your VM called omero-vm-2 which you can make alterations to. This means that you can always return to the original omero-vm if you break anything. From now on **only** make changes to omero-vm-2.

Extending the HDD

By default, your virtual hard-drive attached to omero-vm-2 is of a type which cannot be extended; so you need to change this by cloning your HDD from the VDMK type to VDI type:

```
$ VBoxManage clonehd omero-vm-2-disk1.vmdk omero-vm-2-disk1.vdi --format VDI
```

You now need to increase the size of your virtual HDD. The following command resizes the HDD to 60GB but you should select a size to suit the amount of data you plan to store in OMERO:

```
$ VBoxManage modifyhd omero-vm-2-disk1.vdi --resize 60000
```

Adding the extended HDD to the VM clone

You now need to tell VirtualBox to use `omero-vm-2-disk1.vdi` instead of `omero-vm-2-disk1.vmdk` which is currently attached to the VM. Whilst you are on the *Storage* tab you will also attach the Ubuntu ISO that you downloaded earlier to your VM. This will allow you to use the tools that ship with Ubuntu to make changes to the filesystem within your VM.

1. Start VirtualBox and select omero-vm-2 in the VM library.
2. Right-click *Settings* then select the *Storage* tab.
3. Right-click on *omero-vm-2-disk1.vmdk* and select *Remove attachment*.
4. Next to the *SATA Controller* entry, click the *Add Hard Disk* icon with a green plus sign. In the pop-up dialog, select *Choose existing disk*. Now navigate to the location where VirtualBox stores your virtual machines and enter the `omero-vm-2` directory. Select the `omero-vm-2 disk1.vdi` and click *open*.
5. Add an *IDE Controller* using the *Add Controller* icon. Select this new controller then click *Add CD/DVD device* followed by *Choose Disk*. Navigate to the location of your Ubuntu ISO, select it and click *OK*.

The storage for your OMERO VM should now look similar to *Virtual Appliance storage settings*.

[9] http://www.ubuntu.com/download/desktop

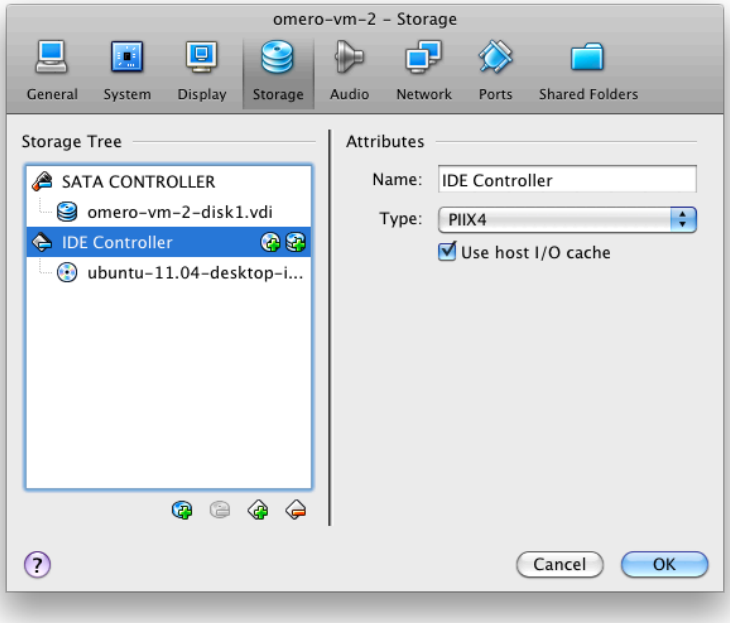


Figure 4.9: Virtual Appliance storage settings

Click *OK* to return to the VirtualBox VM library. With omero-vm-2 selected, ensure that the storage details match what you expect, e.g. omero-vm-2-disk1.vdi is connected to your SATA Port 0. The size for this disk should also more or less match what you specified earlier with the *VBoxManage modifyhd* command. The reported numbers do not exactly matchup, e.g. a virtualised HDD of 60GB size will be reported as 58.59GB.

Reallocating space on the VA HDD

Start the omero-vm-2 VM. Ubuntu linux should boot and you should eventually see a welcome screen giving you the option to try Ubuntu or to install it. You can now start the GParted software and resize your partitions.

1. Select try Ubuntu and you should be presented with a graphical desktop.
2. Start the *gparted* tool using the menu option under *System* → *Administration* → *GParted Partition Editor*.
3. The GParted GUI will display information similar to *Virtual Appliance HDD in GParted*.
4. Right-click the entry for */dev/sda5* and select *Swapoff*.
5. Right click on */dev/sda5* and click *Delete* to remove the swap partition.
6. Delete */dev/sda2* in the same way. This should leave two entries, one for */dev/sda1* and one for unallocated space.
7. Right-click */dev/sda1* and select *Resize*. Now drag the right arrow to the right until the entry for *Free space following (MiB)* is about 2000, then click *Resize/Move*.
8. Right click the entry for unallocated space and select *New* from the pop-up menu. Select *linux-swap* from the *File system* drop-down menu then *Add*.

Up until this point you have not actually applied any of your changes to the HDD, you have only specified a list of changes that should be made. You can now go ahead and apply them by selecting the *Edit* → *Apply All Operations* menu item, then clicking *Apply* in the confirmation dialog box.

When the operations have completed, dismiss the dialog with the *Close* button, close GParted, then shutdown the VM.

Changing HDD settings inside the VA

You no longer need the Ubuntu ISO so you can detach it from your VM. Ensure that omero-vm-2 is selected then click *Settings* and select the *Storage* tab. Right-click the *IDE Controller* entry and select *Remove Controller*, then click *OK* to return the VirtualBox VM library.

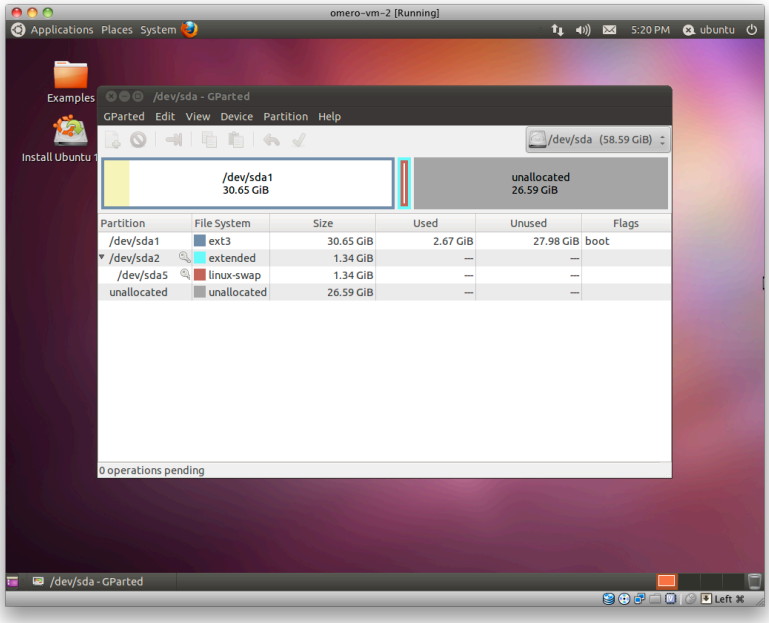


Figure 4.10: Virtual Appliance HDD in GParted

Start the omero-vm-2 VM and allow it to boot. As root then issue the `df -h` command. Verify that the size of the */dev/sda1* is approximately what you expect, e.g. if you allocated a 60GB virtual HDD then after size conversions and swap allocation you should end up with */dev/sda1* reported as being around 56GB.

Within the VM you need to add the UUID of the new swap partition to the `/etc/fstab` file because you deleted the old one and created a new swap meaning that the IDs will no longer match.

```
$ vim /etc/fstab
```

Move your cursor to the entry that looks similar to the following:

```
UUID=SOME-LONG-ALPHA-NUMERIC-STRING none swap sw 0 0
```

then press `i` to enter "insert mode". Delete the alphanumeric string so that the entry looks similar to the following:

```
UUID= none swap sw 0 0
```

and place your cursor after the equals sign. You can now issue a command from within the VIM editor to insert your new swap UUID into the `fstab` file.

```
[Insert Mode] <CTRL-R> =system('/sbin/blkid -t TYPE=swap | cut -c18-53') <return>
```

Save your file and quit VIM:

```
[Command Mode] :wq <return>
```

Now reboot your VM with

```
$ shutdown -r now
```

Once your VM has rebooted you should now have a working VM with a larger virtual HDD.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

4.2 OMERO demo server

Note: This service is no longer available for the 4.4.x series. See the OMERO 5 Demo server page[10] for information on how to apply for an account for the new service.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

[10] http://www.openmicroscopy.org/site/support/omero5/users/demo-server.html

Part II

System Administrator Documentation

Warning: This documentation is for the OMERO 4.4.x line which, on the release of version 5.0 has now entered maintenance mode. While we will continue to support this version throughout 2014, it will only be updated for major bug fixes.

CHAPTER

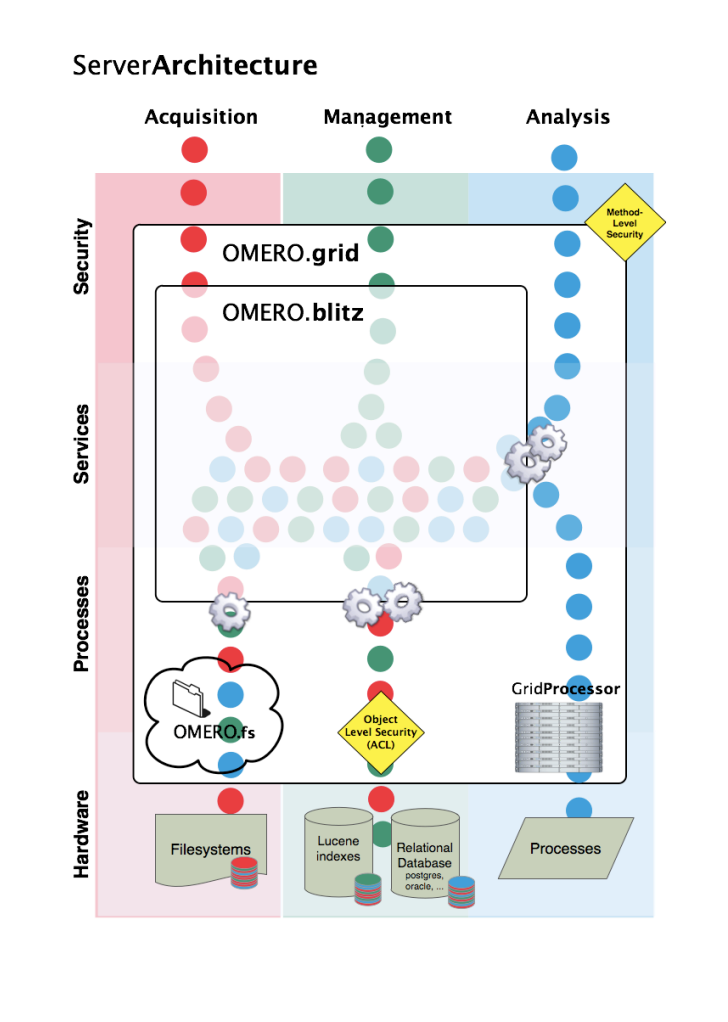
FIVE

SERVER BACKGROUND

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

5.1 Server overview

The OMERO server system provides storage and processing of image data which conforms to the OME Specification[1]. 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.

[1] http://www.openmicroscopy.org/site/support/ome-model/specifications/

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.

5.1.1 Developing the server

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).

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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

5.2 System Requirements

5.2.1 Prerequisites overview

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.editor	Required for some functionality	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	

5.2.2 OMERO.server

The system requirements for OMERO.server vary greatly depending on image size and number of users. At a minimum we suggest:

- Mac OS X 10.5 or later; Windows XP or later; Ubuntu 8.10 or later/Centos 5/Debian Lenny or other Linux distro
- Single core 1.33GHz Intel or AMD CPU
- 2GB RAM
- 500MB of hard drive space for OMERO.server distribution
- Java 1.6 or later
- Python 2.4 or later (2.6 or later on Windows)
- Hard drive space 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 *Unix* and *Windows* binary repository sections for more information.

The recommended OMERO.server specification we suggest for between 25-50 users is:

- Mac OS X 10.5 or later; Windows XP or later; Ubuntu 8.10 or later/Centos 5/Debian Lenny or other Linux distro
- Quad core 1.33GHz Intel or AMD CPU
- 8GB RAM
- 500MB hard drive space for OMERO.server distribution

- Java 1.6 or later
- Python 2.4 or later (2.6 or later on Windows)
- Hard drive space proportional to the image sizes expected (Likely between 10 and 100TB)

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 6GB 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 but I would surely doubt a large deployment using more than a few GBs of RAM for this purpose, it is just not cost effective.

Summary: Depending on hardware layout 16, 24 or 32GB of RAM would be ideal for your OMERO server. If you have a separate database server more than 16GB of RAM may not be of much benefit to you at all.

CPU is really not something that an OMERO system is almost ever limited by. However, when it is limited it is almost always limited by GHz and not by the CPU count. So you are not going to get a huge OMERO experience performance increase by, for example, throwing 24 cores at the problem.

Summary: Depending on hardware layout 2 x 4, 2 x 6 system core count should be more than enough.

5.2.3 OMERO.insight, OMERO.editor and OMERO.importer

- Mac OS X 10.5 or later; Windows XP or later; Ubuntu 8.10 or later/Centos 5/Debian Lenny or other Linux distro
- Single core 1.33GHz Intel or AMD CPU
- 2GB RAM
- 100MB hard drive space for OMERO.clients distribution
- Java 1.6 or later

Large imports may require 4GB RAM.

5.2.4 OpenGL versions of OMERO.insight

- NVIDIA 7xxx or later GPU; ATI 4xxx or later GPU
- 2GB RAM

Note: These are suggested requirements based on limited testing. Your mileage may vary. Any OpenGL 1.3 capable card *should* work.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

5.3 Known Limitations

5.3.1 Data management issues

Moving data between groups

When moving images into a different group from the one they were uploaded into, important attached metadata, such as file annotations and rendering settings, may be lost. This may especially affect moving data into a group with more restrictive permissions. See #10419[2] and #9610[3] for further details and to leave comments on your experience with this problem.

[2] http://trac.openmicroscopy.org.uk/ome/ticket/10419
[3] http://trac.openmicroscopy.org.uk/ome/ticket/9610

Moving projected images

Depending on the group permissions, you may not be able to move projected images nor the original images that they were created from, into a different group (see #11532[4] for a work-around or to comment on your experience). Moving a dataset containing projected images may result in an error when trying to view the projected images, and the original image may be lost. See #10262[5] for further details.

Deleting projected images

It may not be possible to delete an image you own in a collaborative group if a projected image has been created from it, due to shared metadata. See #11529[6] for further details or to comment on your experience with this issue.

Deleting SVS images

Deleting an image which shares an original SVS file with another image is currently resulting in the archived original SVS file also being deleted and no longer accessible from the other image. See #11348[7] for further details or to comment on your experience with this problem.

5.3.2 Windows OS issues

OMERO.web 'development server' does not work under Windows XP

The development server included with OMERO.web does not work under Windows XP. This server is included to allow you to easily evaluate and develop the OMERO.web component. As it should not be used in a production environment, while an inconvenience, this should not stop the deployment of a production version of OMERO.web. See OME forum topic[8].

Binary delete

On Windows servers not all binary files corresponding to a delete may be removed from the binary repository. See *Binary data* for more details.

Ice 3.5

The Ice 3.5 OMERO.server is not supported on Windows out of the box because the Ice 3.5 binaries from ZeroC[9] require Python 3, which OMERO does not support (see OME forum topic[10]). This limitation does not affect Windows Ice 3.5 clients which can connect to an Ice 3.4 server.

5.3.3 Mac OS X issues

C++ compilation problems on Mac OS X 10.6 (Snow Leopard)

Under certain circumstances building OmeroCpp can fail with "ld: symbol(s) not found". You can find further details, a potential solution and make any comments on your experience with the problem on #3210[11].

[4] http://trac.openmicroscopy.org.uk/ome/ticket/11532
[5] http://trac.openmicroscopy.org.uk/ome/ticket/10262
[6] http://trac.openmicroscopy.org.uk/ome/ticket/11529
[7] http://trac.openmicroscopy.org.uk/ome/ticket/11348
[8] http://www.openmicroscopy.org/community/viewtopic.php?f=5&t=640
[9] http://www.zeroc.com
[10] http://www.openmicroscopy.org/community/viewtopic.php?f=5&t=7352
[11] http://trac.openmicroscopy.org.uk/ome/ticket/3210

5.3.4 Ubuntu issues

Importing using desktop clients

Under older Ubuntu installations, the 'import folder' option in the desktop clients currently does not work.

5.3.5 Searching

Not finding certain pieces of data in wildcard searches

Sometimes data is missed when certain types of wildcard searches are performed. You can find further details on #3164[12].

5.3.6 File format support

DeltaVision OMX files (.hdr) files not viewable

Some files generated by an OMX system have a pixel type of float and a large dynamic range. While the files import into OMERO they are currently not viewable. This fix would require deep changes to several parts of the code and we have chosen not to make the changes yet. See #3256[13].

5.3.7 LDAP

Enabling synchronization of LDAP on user login will result in LDAP being treated as the authority on both group membership and also the available groups. Any groups defined in OMERO and not in LDAP will result in users being removed from these groups. The groups will still exist in OMERO but user membership will be treated as being defined by LDAP alone.

[12] http://trac.openmicroscopy.org.uk/ome/ticket/3164
[13] http://trac.openmicroscopy.org.uk/ome/ticket/3256

CHAPTER

SIX

BASIC UNIX SERVER INSTALLATION

This chapter contains the instructions to install OMERO.server on UNIX & UNIX-like platforms.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

6.1 OMERO.server installation

See also:

OMERO.server upgrade Instructions for **upgrading** your OMERO.server installation.

OMERO.server Mac OS X installation walk-through with Homebrew Instructions for installing your OMERO.server on Mac OS X using Homebrew.

OMERO.server Linux installation walk-through Instructions for installing your OMERO.server on Debian/Ubuntu.

OMERO Installation for Mac OS X Snow Leopard (10.6)[1] A guide provided by Janek Claus and Kenneth Arcieri (NIH/NICHD/UCSS) for users wishing to install OMERO 4.1 on Mac OS 10.6, including details on how to build all dependencies from source. The OMERO 4.4 and 4.3 installs are almost identical to 4.2 and 4.1 (except for web client), so this guide will still be useful.

Omero Server Beta 4.2.0 installation on CentOS[2] A guide provided by Caterina Strambio De Castillia and Vanni Galli (University of Geneva/SUPSI in Lugano) for users wishing to install OMERO 4.1.1 or 4.2.0 on CentOS.

6.1.1 Limitations

Installation **will require a "root" level account** for which you know the password. 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?*.

6.1.2 Prerequisites

Note: The installation of these prerequisite applications is outside the scope of this document. For Linux distributions you should use the default package manager.

The following are necessary:

Java SE Development Kit (JDK) (1.6 or higher)

Java SE Downloads are available from http://www.oracle.com/technetwork/java/javase/downloads/index.html

UNIX systems, you can check which version of Java is currently available to you via your $PATH as follows:

```
$ which java
/usr/bin/java
$ java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) Server VM (build 1.6.0-b105, mixed mode)
```

Further, you can see if you have the Java compiler (Java SDK) installed and available via your $PATH as follows…

```
$ which javac
/usr/bin/javac
$ javac -version
javac 1.6.0
```

Python 2 (2.4 or higher)

Check you have Python (and what version) by typing:

```
$ python --version
Python 2.5.1
```

OMERO does not support Python 3.

The following are optional depending on what services you require:

Package	Functionality	Downloads
Python Imaging Library[14] [15]	OMERO.web and Figure Export	PIL page[16]
Matplotlib[17]	OMERO.web	Matplotlib page[18]
NumPy (1.2.0 or higher) [19]	Scripting	Numpy/Scipy page[20]
PyTables (2.1.0 or higher)	*OMERO.tables*	PyTables page[21]
scipy.ndimage	Volume Viewer[22] [23]	Numpy/Scipy page[24]

Ice (3.3.x or 3.4.x)

Note: OMERO 4.4 supports Ice3.4, but this requires the correct version of OMERO.server (see Downloads[25]). See "Do I use Ice 3.3 or Ice 3.4?"[26] in the FAQ. If you have further questions, consult the Forums[27].

[3] http://www.pythonware.com/products/pil/
[4] Make sure to have libjpeg (http://libjpeg.sourceforge.net/) installed when building the Python Imaging Library (http://www.pythonware.com/products/pil/).
[5] http://www.pythonware.com/products/pil/
[6] http://matplotlib.org/
[7] http://matplotlib.org/
[8] May already have been installed as a dependency of Matplot Lib.
[9] http://www.scipy.org/Download
[10] http://www.pytables.org/moin/Downloads
[11] http://www.openmicroscopy.org/site/products/omero/volume-viewer-in-omero.web
[12] Allows larger volumes to be viewed in the Volume Viewer (http://www.openmicroscopy.org/site/products/omero/volume-viewer-in-omero.web).
[13] http://www.scipy.org/Download
[14] http://www.pythonware.com/products/pil/
[15] Make sure to have libjpeg (http://libjpeg.sourceforge.net/) installed when building the Python Imaging Library (http://www.pythonware.com/products/pil/).
[16] http://www.pythonware.com/products/pil/
[17] http://matplotlib.org/
[18] http://matplotlib.org/
[19] May already have been installed as a dependency of Matplot Lib.
[20] http://www.scipy.org/Download
[21] http://www.pytables.org/moin/Downloads
[22] http://www.openmicroscopy.org/site/products/omero/volume-viewer-in-omero.web
[23] Allows larger volumes to be viewed in the Volume Viewer (http://www.openmicroscopy.org/site/products/omero/volume-viewer-in-omero.web).
[24] http://www.scipy.org/Download
[25] http://downloads.openmicroscopy.org/latest/omero4/
[26] http://www.openmicroscopy.org/site/support/faq/omero/do-i-use-ice-3.3-or-ice-3.4
[27] http://www.openmicroscopy.org/community/

UNIX source downloads and binary packages are available from ZeroC[28]. The latest compatible distribution is the 3.3.1 release[29]. ZeroC does not provide binaries which work out of the box for Mac OS X Snow Leopard (10.6), so we recommend you follow the instructions for installation using *Homebrew* instead if this is your OS.

PostgreSQL (8.4 or higher)

PostgreSQL installed and configured with PL/pgSQL and to accept TCP connections. PostgreSQL 8.3 and earlier are not supported. See *OMERO.server and PostgreSQL* for specifics about each version.

OMERO.server

The `OMERO.server.zip` is available from the OMERO downloads[30] page.

6.1.3 Environment variables

For the prerequisite software to run properly, your PATH, PYTHONPATH, and (DY)LD_LIBRARY_PATH environment variables must be configured. If you installed via a package manager such as rpm, apt-get, or macports, they should be set for you. If not correctly configured or if you installed manually to /opt/Ice-… or a similar location, you will need to set the values yourself.

If you are running a Linux distribution such as Debian or Ubuntu and have used APT to install the prerequisites then the installed software will have been installed to locations in your file system according to the Debian Policy Manual for software packaging. You can explicitly set your environment variables to reflect these install locations by editing the .bashrc (if on Linux) or .profile (if on Mac OS X) file which can be found within your home directory. For example, as of writing, on Debian and Ubuntu the following environment variables should be set:

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun
export JRE_HOME=/usr/lib/jvm/java-6-sun
export ICE_HOME=/usr/share/Ice-3.3.1
export POSTGRES_HOME=/usr/lib/postgresql/8.4
export PYTHONPATH=/usr/lib/pymodules/python2.6:$PYTHONPATH
export DYLD_LIBRARY_PATH=/usr/share/java:/usr/lib/:$DYLD_LIBRARY_PATH
export LD_LIBRARY_PATH=/usr/share/java:/usr/lib:$LD_LIBRARY_PATH
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin:$ICE_HOME/bin:$POSTGRES_HOME/bin
```

Please note that the precise details of these environment variables can change as new versions of software are released. You can retrieve the pathname for a file by using the which command. So if you are unsure what path to use in your environment variables, e.g. for the ICE_HOME variable you can execute the following command:

```
$ which icegridnode
/Users/ome/apps/OMERO.libs/bin/icegridnode
```

You can now set the ICE_HOME path to something similar to **/Users/ome/apps/OMERO.libs/bin** based upon the output from which, e.g.

```
export ICE_HOME=/Users/ome/apps/OMERO.libs/bin/icegridnode
```

As a last ditch effort, on a Linux or Mac OS X machine you can use the find command to help you discover whereabouts something is located in your filesystem. e.g.

```
$ find / -name "icegridnode" 2>/dev/null
```

[28] http://www.zeroc.com
[29] http://zeroc.com/download_3_3_1.html
[30] http://downloads.openmicroscopy.org/latest/omero4/

However this might take a long time to run, especially on a big filesystem, so you might get a more timely solution by going to the OMERO forums.

If the command gives no output then perhaps Ice is not installed, in which case you should see the section above on installing Ice.

You can also add your OMERO bin directory to your path like so:

```
export PATH=$PATH:path-to-your-omero-install-directory/bin
```

When performing some operations the clients make use of temporary file storage and log directories. By default these files are stored below the users HOME directory in `$HOME/omero/tmp`, `$HOME/omero/log` and `$HOME/omero/sessions` (resp. `$HOME\omero\tmp`, `$HOME\omero\log` and `$HOME\omero\sessions`). If your home(~) directory `$HOME` 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.

The OMERO.server also access the `$HOME/omero/tmp` and `$HOME/omero/log` folders of **the user the server process is running as** (resp. `$HOME\omero\tmp` and `$HOME\omero\log`). As the server makes heavier use of these folders than the clients, if the users home(~) is stored on a network the server can be slowed down. To get round this for the OMERO.server you can define an `OMERO_TEMPDIR` environment variable pointing to a temporary directory located on the local file system e.g. `/tmp` (resp. `C:\tmp\`).

6.1.4 Creating a database as root

Probably the most important step towards having a working server system is having a properly configured 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 below:

- Create a non-superuser database user and record the name and password. You will need to configure OMERO to use your username and password by setting the omero.db.name and omero.db.pass properties (below).

```
# For PostgreSQL 8.4.x and later
$ sudo -u postgres createuser -P -D -R -S db_user
Enter password for new role:       # db_password
Enter it again:       # db_password
```

> **Warning:** For illustrative purposes, the default name and password for the user are `db_user` and `db_password` respectively. However, you should not use these default values for your installation but use your own choice of username and password instead.

- Create a database for OMERO to reside in

```
$ sudo -u postgres createdb -O db_user omero_database
```

- Add the PL/pgSQL language to your database

```
$ sudo -u postgres createlang plpgsql 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.

6.1.5 Location for the your OMERO binary repository

- Create a directory for the OMERO binary data repository. `/OMERO` (resp. `C:\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 that 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
ome
$ sudo chown -R ome /OMERO
```

6.1.6 Installation

- Extract the OMERO tarball and note its location. Below it is referred to as: `~/Desktop/omero`
- Optionally, review `~/Desktop/omero/etc/omero.properties` which contains all default settings.

  You will need to open the file with a text editor. Do not edit the file. Any configuration settings you would like to change can be changed in the next step.
- 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 link[31])

```
$ cd ~/Desktop/omero
$ bin/omero config set omero.db.name 'omero_database'
$ bin/omero config set omero.db.user 'db_user'
$ bin/omero config set omero.db.pass 'db_password'
```

- If you have chosen a non-standard *OMERO binary repository* location above, be sure to configure the `omero.data.dir` property.
- Create the OMERO database initialization script. You will be asked for the version of the script which you would like to generate, where both defaults can be accepted. Finally, you will be asked to enter and confirm password for your newly created OMERO root user.

> **Warning:** For illustrative purposes, the default password for the OMERO rootuser is `root_password`. However, you should not use this default value for your installation but use your own choice of password instead.
> This should **not** be the same password as your Linux/Mac/Windows root user!

[31] http://www.openmicroscopy.org/community/viewtopic.php?f=5&t=360#p922

```
$ cd ~/Desktop/omero
$ bin/omero db script
Please enter omero.db.version [OMERO4.4]:
Please enter omero.db.patch [0]:
Please enter password for new OMERO root user: # root_password
Please re-enter password for new OMERO root user: # root_password
Saving to ~/Desktop/omero/OMERO4.4__0.sql
```

- Initialize your database with the script.

```
$ psql -h localhost -U db_user omero_database < OMERO4.4__0.sql
```

- Before starting the OMERO.server we should run the OMERO diagnostics script so that we check that all of our settings are correct, e.g.

```
$ bin/omero admin diagnostics
```

- You can now start the server using:

```
$ bin/omero admin start
Creating var/master
Initializing var/log
Creating var/registry
No descriptor given. Using etc/grid/default.xml
```

- If you would like to move the directory again, see `bin\winconfig.bat --help`, which gets called automatically on an initial install.
- You can now test that you can log-in as "root", either with the OMERO.insight client or command-line:

```
$ bin/omero login
Server: [localhost]
Username: [root]
Password:         # root_password
```

6.1.7 OMERO.web and administration

Note: In order to deploy OMERO.web in a production environment such as Apache or IIS please follow the instructions under *OMERO.web deployment*.

Otherwise, the internal Django webserver can be used for evaluation and development. In this case, we need to turn debugging on, in order that static files can be served by Django:

```
$ bin/omero config set omero.web.application_server development
$ bin/omero config set omero.web.debug True
```

then start by

```
$ bin/omero web start
Starting django development webserver...
Validating models...
0 errors found

Django version 1.1.1, using settings 'omeroweb.settings'
Development server is running at http://0.0.0.0:4080/
Quit the server with CONTROL-C.
```

Once you have deployed and started the server you can use your browser to access the OMERO.web interface:

- http://localhost:4080/

Figure 6.1: OMERO.webadmin login

6.1.8 Enabling movie creation from OMERO.

OMERO has the facility to create AVI/MPEG Movies from Images, which can be called from OMERO.insight. The page *OMERO.movie* gives details on how to enable them.

6.1.9 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[32]. See the Feature list[33] for more advanced features you may want to use, and *Advanced configuration* on how to get the most out of your server.

JVM memory settings

The most likely change you will need to make to your application descriptors is increasing the memory settings. This is not done by default since it would prevent starting the server on some sites' test instance, but for production use a setting higher than 512MB is **highly** recommended.

You can either edit the file manually, or use a small script such as:

[32] http://cvs.openmicroscopy.org.uk/snapshots/movies/omero-4-1/mov/FlexPreview4.1-configuration.mov
[33] http://www.openmicroscopy.org/site/products/omero/feature-list

```
perl -i -pe 's/Xmx512M/Xmx2048M/' etc/grid/templates.xml
perl -i -pe 's/XX:MaxPermSize=128m/XX:MaxPermSize=256M/' etc/grid/templates.xml
```

See the grid configuration section in the *Advanced server configuration* documentation for more information on *grid/templates.xml*.

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.

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 our ome-users[34] mailing list discussed on the *Community support* page.

OMERO diagnostics

If you want help with your server installation, please include the output of the diagnostics command:

```
$ bin/omero admin diagnostics

================================================================================
OMERO Diagnostics 4.4.12
================================================================================

Commands:   java -version                   1.6.0     (/usr/bin/java)
Commands:   python -V                       2.6.5     (/usr/bin/python)
Commands:   icegridnode --version           3.3.1     (/usr/bin/icegridnode)
Commands:   icegridadmin --version          3.3.1     (/usr/bin/icegridadmin)
Commands:   psql --version                  8.4.12    (/usr/bin/psql)

Server:     icegridnode                     running
Server:     Blitz-0                         active (pid = 28933, enabled)
Server:     DropBox                         active (pid = 28951, enabled)
Server:     FileServer                      active (pid = 28954, enabled)
Server:     Indexer-0                       active (pid = 28957, enabled)
Server:     MonitorServer                   active (pid = 28960, enabled)
Server:     OMERO.Glacier2                  active (pid = 28962, enabled)
Server:     OMERO.IceStorm                  active (pid = 28964, enabled)
Server:     PixelData-0                     active (pid = 28963, enabled)
Server:     Processor-0                     active (pid = 28972, enabled)
Server:     Tables-0                        active (pid = 28974, enabled)
Server:     TestDropBox                     inactive (enabled)

Log dir:    /home/omero/OMERO.server-4.4

Log files:  Blitz-0.log                 360.0 MB      errors=9    warnings=2458
Log files:  DropBox.log                 3.0 KB        errors=0    warnings=1
Log files:  FileServer.log              0.0 KB
Log files:  Indexer-0.log               506.0 KB      errors=0    warnings=90
Log files:  MonitorServer.log           2.0 KB
Log files:  OMEROweb.log                710.0 KB      errors=5    warnings=2
Log files:  OMEROweb.log.1              777.0 KB      errors=0    warnings=1
Log files:  OMEROweb.log.2              776.0 KB      errors=0    warnings=2
Log files:  OMEROweb.log.3              777.0 KB
```

[34] http://lists.openmicroscopy.org.uk/mailman/listinfo/ome-users

```
Log files:  OMEROweb.log.4                  879.0 KB      errors=1    warnings=2
Log files:  OMEROweb.log.5                  258.0 KB
Log files:  OMEROweb_request.log            10.0 KB       errors=3    warnings=3
Log files:  PixelData-0.log                 4.0 KB
Log files:  Processor-0.log                 315.0 KB      errors=0    warnings=1
Log files:  Tables-0.log                    2.0 KB        errors=0    warnings=1
Log files:  TestDropBox.log                 n/a
Log files:  master.err                      0.0 KB
Log files:  master.out                      0.0 KB
Log files:  Total size                      365.49 MB

Parsing Blitz-0.log:[line:30] => Server restarted <=
Parsing Blitz-0.log:[line:213] => Server restarted <=

Environment:OMERO_HOME=(unset)
Environment:OMERO_NODE=(unset)
Environment:OMERO_MASTER=(unset)
Environment:PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
Environment:ICE_HOME=(unset)
Environment:LD_LIBRARY_PATH=(unset)
Environment:DYLD_LIBRARY_PATH=(unset)

OMERO data dir: '/OMERO'    Exists? True    Is writable? True
OMERO.web status... [RUNNING] (PID 28736)
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

6.2 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. Its structure is based on OMEIS[a].

[a] http://www.openmicroscopy.org/site/support/previous/ome-server/system-overview/ome-image-server/

6.2.1 Layout

The repository is internally laid out as follows:

```
/OMERO
/OMERO/Pixels      <--- Pixel data
/OMERO/Files       <--- Original file data
/OMERO/Thumbnails  <--- Thumbnail data
/OMERO/FullText    <--- Lucene full text search index
```

Your repository is not:

- the "database"
- the directory where your OMERO.server binaries are
- the directory where your OMERO.client (OMERO.insight, OMERO.editor or OMERO.importer) binaries are
- your PostgreSQL data directory

6.2.2 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 NullPointerExceptions, "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.

6.2.3 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:

```
$ cd OMERO.server
$ bin/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:

```
$ cd OMERO.server
$ bin/omero admin stop
$ mv /OMERO /mnt/really_big_disk
$ bin/omero config set omero.data.dir /mnt/really_big_disk/OMERO
$ bin/omero admin start
```

6.2.4 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  2 omero  omero  1656 Dec 18 14:31 Files
drwxr-xr-x 25 omero  omero 23256 Dec 10 19:06 Pixels
drwxr-xr-x  2 omero  omero    48 Dec  8  2006 Thumbnails
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

6.3 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.

6.3.1 Ensuring you have a valid PostgreSQL version

For OMERO 4.4, PostgreSQL 8.4 or higher is required.

You can check which version of PostgreSQL you have installed with any of the following commands:

```
$ createuser -V
createuser (PostgreSQL) 9.1.4
$ psql -V
psql (PostgreSQL) 9.1.4
$ createdb -V
createdb (PostgreSQL) 9.1.4
```

If your default PostgreSQL installation is version 8.3 or earlier, you will need to upgrade to a more up-to-date version. We suggest the installer from EnterpriseDB[35]. Versions 8.4, 9.0 and 9.1 are known to work with OMERO 4.4; 9.1 is recommended.

Compatibility matrix

Versions of PostgreSQL which are compatible with OMERO are shown in the table below.

PostgreSQL	OMERO 4.1	OMERO 4.2	OMERO 4.3	OMERO 4.4
7.4	YES	NO *[1]*	NO *[1]*	NO *[4]*
8.1	YES	NO *[3]*	NO *[1]*	NO *[4]*
8.2	YES	YES	NO *[3]*	NO *[4]*
8.3	YES	YES	YES	NO *[4]*
8.4	YES	YES	YES	YES
9.x	YES *[2]*	YES *[2]*	YES *[2]*	YES

[1] Not suggested; see #4902[36]

[2] Configuration may be necessary; see #5662[37]

[3] Not suggested; see #5861[38]

[4] Unsupported; see #7813[39]

6.3.2 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`:

[35] http://www.enterprisedb.com/
[36] http://trac.openmicroscopy.org.uk/ome/ticket/4902
[37] http://trac.openmicroscopy.org.uk/ome/ticket/5662
[38] http://trac.openmicroscopy.org.uk/ome/ticket/5861
[39] http://trac.openmicroscopy.org.uk/ome/ticket/7813

```
$ 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. For PostgreSQL 8.4 and 9.x, 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
```

6.3.3 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
```

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**.

See also:

PostgreSQL[40] Interactive documentation for the current release of PostgreSQL.

Connections and Authentication[41] Section of the PostgreSQL documentation about configuring the server using *postgresql.conf*.

Client Authentication[42] Chapter of the PostgreSQL documentation about configuring client authentication with *pg_hba.conf*.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

6.4 OMERO.server Mac OS X installation walk-through with Homebrew

Overview

This walk-through is a list of the commands used to install OMERO on a clean Mac OS X 10.7 Lion using Homebrew.

The instructions provided here depend on Homebrew 0.9 or later, including support for the `brew tap` command. These instructions are implemented in an automated script[43] which installs OMERO via Homebrew from a fresh configuration.

6.4.1 Prerequisites

OS X/Xcode

Install the OS X Developer Tools[44]. This procedure is regularly tested with the following configuration:

Model Identifier	Mac OS X version	Xcode version
MacBookPro8,2 (Intel Core i7, 2.3 GHz, 8 GB RAM)	10.7.5	4.6

Note: For Xcode 4.x, make sure that the Command line tools are installed (*Preferences* → *Downloads* → *Components*)

Java (>=1.6)

You need Java which comes as standard on OS X.

```
$ java -version
java version "1.6.0_33"
Java(TM) SE Runtime Environment (build 1.6.0_33-b03-424-11M3720)
Java HotSpot(TM) 64-Bit Server VM (build 20.8-b03-424, mixed mode)
```

Homebrew

Follow the installation instructions on the Homebrew wiki[45]. All requirements for OMERO will be installed to `/usr/local`.

```
$ ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go/install)"
$ brew install git
```

If you are having issues with curl, see the *curl (Mac 10.5 only)* section under *Common issues*.

The installation of OMERO via Homebrew depends on two alternate repositories containing extra formulas: https://github.com/Homebrew/homebrew-science for the HDF5 formula and https://github.com/ome/homebrew-alt for all the OME-provided formulas and older versions of Ice.

Python (>=2.4)

You can install OMERO using either the system-wide Python or the Python provided by Homebrew. For a more thorough description of the latter solution, look at the Homebrew and Python[46] page. Note that the automated script linked above tests the OMERO installation using the Homebrew Python.

If using system-wide Python, check it is installed and its version.

```
$ python --version
Python 2.7.3
```

To install the Python provided by Homebrew:

```
$ brew install python
```

[43] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/docs/hudson/OMERO-homebrew-install.sh
[44] https://developer.apple.com/technologies/tools/
[45] https://github.com/mxcl/homebrew/wiki/installation
[46] https://github.com/mxcl/homebrew/wiki/Homebrew-and-Python

Independently of the chosen Python, you can setup and use virtual environments[47] to install the OMERO Python dependencies (see *Python dependencies*).

Note: The Homebrew formulas used below provide Python bindings. As described in Homebrew and Python[48], you should NOT be in an active virtual environment when you `brew install` them.

6.4.2 OMERO installation

OMERO 4.4.12

If you just want a deployment of the 4.4.12 release of OMERO.server then a simple Homebrew install is sufficient, e.g.

```
$ brew tap homebrew/science
$ brew tap ome/alt
$ brew install omero
```

This should install OMERO along with most of the non-Python requirements.

The default version of Ice installed by the OMERO formula is Ice 3.5. To install OMERO with Ice 3.4, use:

```
$ brew install omero --with-ice34
```

or to install OMERO with Ice 3.3, use:

```
$ brew install omero --with-ice33
```

Additional installation options can be listed using the `info` command:

```
$ brew info omero
```

Development server

If you wish to pull OMERO.server from the git repo for development purposes then it is worth setting up OMERO.server manually. First use Homebrew to install the OMERO dependencies:

```
$ brew tap homebrew/science
$ brew tap ome/alt
$ brew install `brew deps omero`
```

The default version of Ice installed by the OMERO formula is Ice 3.5. To install the OMERO dependencies with Ice 3.4, use:

```
$ brew install `brew deps omero --with-ice34`
```

or to install the OMERO dependencies with Ice 3.3, use:

```
$ brew install `brew deps omero --with-ice33`
```

Prepare a place for your OMERO code to live, e.g.

[47] http://www.virtualenv.org/en/latest/
[48] https://github.com/mxcl/homebrew/wiki/Homebrew-and-Python

```
$ mkdir -p ~/code/projects/OMERO
$ cd ~/code/projects/OMERO
```

If you installed Ice 3.5, you will need to set `SLICEPATH` to be able to build the server, i.e. `export SLICEPATH=/usr/local/share/Ice-3.5/slice`.

If you want the development version of OMERO.server, you can clone the source code from the project's GitHub account to build locally:

```
$ git clone --recursive git://github.com/openmicroscopy/openmicroscopy
$ cd openmicroscopy && ./build.py
```

Note: If you have a GitHub account and you plan to develop code for OMERO, you should make a fork into your own account and then clone this fork to your local development machine, e.g.

```
$ git clone --recursive git://github.com/YOURNAMEHERE/openmicroscopy
$ cd openmicroscopy && ./build.py
```

See also:

Checking out the source code Developer documentation page on how to check out to source code

Build System Developer documentation page on how to build the OMERO.server

Alternatively, you can download a daily build[49] of the OMERO.server from our continuous integration server.

6.4.3 Additional OMERO requirements

PostgreSQL

Install PostgreSQL if you do not have another PostgreSQL installation that you can use.

```
$ brew install postgresql
```

Python dependencies

The Python dependencies can be installed in the system-wide Python site-packages, in the Homebrew Python site-packages or within a virtual environment. If you are using the system-wide Python site-packages, you may need to use `sudo` to install the dependencies. If you are using a virtual environment, activate it before calling the Python dependencies installation script.

If you installed OMERO using Homebrew, execute the `omero_python_deps` script:

```
$ cd /usr/local
$ bash bin/omero_python_deps
```

If you use a development server, execute the `python_deps.sh` script under `docs/install`:

```
$ cd ~/code/projects/OMERO
$ bash docs/install/python_deps.sh
```

If you encounter problems with the installation script, please take a look at *Common issues*.

[49] http://ci.openmicroscopy.org/job/OMERO-trunk/lastSuccessfulBuild/artifact/

6.4.4 Configuration

Environment variables

Edit your `.profile` as appropriate. The following are indicators of required entries and correspond to a Homebrew installation of OMERO 4.4:

```
export ICE_CONFIG=$(brew --prefix omero)/etc/ice.config
export ICE_HOME=$(brew --prefix ice)
export PYTHONPATH=$(brew --prefix omero)/lib/python:/usr/local/lib/python2.7/site-packages

export PATH=/usr/local/bin:/usr/local/sbin:/usr/local/lib/node_modules:$ICE_HOME/bin:$PATH
export DYLD_LIBRARY_PATH=$ICE_HOME/lib:$DYLD_LIBRARY_PATH
```

If you have installed Ice 3.3, replace `ICE_HOME` by `$(brew --prefix zeroc-ice33)`. If you have installed Ice 3.4, replace `ICE_HOME` by `$(brew --prefix zeroc-ice34)`. For both Ice 3.3 and Ice 3.4, use `export PYTHONPATH=$(brew --prefix omero)/lib/python:$ICE_HOME/python`.

Note: If you have a local `.bash_profile` file, it will override your `.profile` configuration file.

Note: On Mac OS X Lion, a version of PostgreSQL is already installed. If you get an error like the following:

```
psql: could not connect to server: Permission denied
Is the server running locally and accepting
connections on Unix domain socket "/var/pgsql_socket/.s.PGSQL.5432"?
```

make sure `/usr/local/bin` is at the beginning of your `PATH` (see also this post[50]).

Database creation

Start the PostgreSQL server.

```
$ initdb /usr/local/var/postgres
$ brew services start postgresql
$ pg_ctl -D /usr/local/var/postgres/ -l /usr/local/var/postgres/server.log start
```

Create a user, a database and add the PL/pgSQL language to your database.

```
$ createuser -P -D -R -S db_user
Enter password for new role:       # db_password
Enter it again:       # db_password
$ createdb -O db_user omero_database
$ createlang plpgsql omero_database
```

Check to make sure the database has been created.

```
$ psql -h localhost -U db_user -l
```

This command should give similar output to the following:

[50] http://nextmarvel.net/blog/2011/09/brew-install-postgresql-on-os-x-lion/

```
                        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           +
                |         |          |             |             | ome=CTc/ome
 template1      | ome     | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 | =c/ome           +
                |         |          |             |             | ome=CTc/ome
(4 rows)
```

OMERO.server

Now tell OMERO.server about our database.

```
$ omero config set omero.db.name omero_database
$ omero config set omero.db.user db_user
$ omero config set omero.db.pass db_password

$ omero db script
Please enter omero.db.version [OMERO4.4]:
Please enter omero.db.patch [0]:
Please enter password for new OMERO root user:       # root_password
Please re-enter password for new OMERO root user:      # root_password
Saving to ~/OMERO4.4__0.sql
```

Then enter the name of the .sql (see last line above) in the next command, to create the database:

```
$ psql -h localhost -U db_user omero_database < OMERO4.4__0.sql
```

Now create a location to store OMERO data, e.g.

```
$ mkdir -p ~/var/OMERO.data
```

and tell OMERO.server this location:

```
$ omero config set omero.data.dir ~/var/OMERO.data
```

We can inspect the OMERO.server configuration settings using:

```
$ omero config get
```

Now start the OMERO.server

```
$ omero admin start
```

Now connect to your OMERO.server using OMERO.insight with the following credentials:

```
U: root
P: root_password
```

OMERO.web

You can set up the internal development web server

```
$ omero config set omero.web.application_server development
$ omero config set omero.web.debug True
```

Then start the webserver with:

```
$ omero web start
Starting django development webserver...
Validating models...
0 errors found

Django version 1.1.1, using settings 'omeroweb.settings'
Development server is running at http://0.0.0.0:4080/
Quit the server with CONTROL-C.
```

6.4.5 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[51].

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.

curl (Mac 10.5 only)

```
curl: (60) SSL certificate problem, verify that the CA cert is OK. Details:
error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
```

Use `export GIT_SSL_NO_VERIFY=1` before running failing brew commands.

Xcode

```
Warning: Xcode is not installed! Builds may fail!
```

Install Xcode using Mac App store[52].

Macports/Fink

```
Warning: It appears you have MacPorts or Fink installed.
```

Follow uninstall instructions from the Macports guide[53].

[51] https://github.com/mxcl/homebrew/wiki/Bug-Fixing-Checklist
[52] https://developer.apple.com/technologies/tools/
[53] http://guide.macports.org/chunked/installing.macports.uninstalling.html

PostgreSQL

```
==> Installing postgresql dependency: readline
Error: No such file or directory - /usr/bin/cc
```

For Xcode 4.3.2 make sure Xcode Command Line Tools are installed (see comment[54]).

```
Error: You must ``brew link ossp-uuid' before postgresql can be installed
```

Try:

```
$ brew cleanup
$ brew link ossp-uuid
```

Ice

```
Error: Failed executing: cd cpp && make M PP_HOME=/Users/sebastien/apps/    OMERO.libs/Cellar/mcpp/2.7.
```

We have had problems building `zeroc-ice33` under MacOS 10.7.3 and 10.6.8 (see #8075[55]). You can try installing `zeroc-ice34` (Ice 3.4) instead. If you decide to go with `zeroc-ice33`, make sure that you do not have `DYLD_LIBRARY_PATH` set to an existing Ice's installation lib directory path. In essence your `.bash_profile` shouldn't have any OMERO-related environment variables set before executing the installation script.

szip

```
==> 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.)
```

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 a too low version number, verify that you have not before installed any Python packages using pip. In the case where pip has been installed before homebrew, uninstall it:

```
$ sudo pip uninstall pip
```

After that try running `python_deps.sh` again. That should install `pip` via Homebrew and put the Python packages in correct folders.

[54] https://github.com/mxcl/homebrew/issues/10244#issuecomment-4013781
[55] http://trac.openmicroscopy.org.uk/ome/ticket/8075

gfortran

`gfortran` currently fails to build on 32-bit 10.6.8 machines (see https://github.com/mxcl/homebrew/issues/17776)

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

6.5 OMERO.server Linux installation walk-through

This page walks through the process of installing OMERO.server on a machine using a Debian-based Linux distribution.

Note: This page is generally applicable to Debian and Ubuntu installations, although there are some small differences which are noted when applicable during the walk-through.

6.5.1 Distributions

Whilst OMERO can be made to work on a wide range of Linux distributions, installation using a package manager is the most straightforward way to get an OMERO installation up and running. However, due to changes between releases of Ubuntu and Debian, there are some restrictions over which version of OMERO can be easily installed using the package manager to install and manage the OMERO prerequisites.

Distribution	ZeroC Ice version	OMERO version
Debian 7.0	3.4	4.4.x
Debian 6.0	3.3	4.3.x, 4.4.x
Ubuntu 12.04 (LTS)	3.4	4.4.x
Ubuntu 11.10	3.4	4.4.x
Ubuntu 11.04	3.3	4.3.x, 4.4.x
Ubuntu 10.04 (LTS)	3.3	4.3.x, 4.4.x

Note: ZeroC[56] Ice can always be built from source code for specific platforms.

In the remainder of this guide you should adjust version numbers to suit the distribution that you are targeting.

6.5.2 Prerequisites

You require at least a clean minimal Debian or Ubuntu installation and a non-root user account that has sudo privileges.

First you need to enable the contrib and non-free repositories by opening `/etc/apt/sources.list` in an editor, e.g. **$ sudo vim /etc/apt/sources.list** and editing to add the following lines:

```
deb http://ftp.uk.debian.org/debian/ squeeze contrib
deb-src http://ftp.uk.debian.org/debian/ squeeze contrib

deb http://ftp.uk.debian.org/debian/ squeeze non-free
deb-src http://ftp.uk.debian.org/debian/ squeeze non-free
```

Note: For Ubuntu the repository names and locations are different to Debian but you need to enable the **main**, **restricted**, **universe** and **multiverse** repositories. You can do this either by editing `/etc/apt/sources.list` directly, in which case the entries already exist but are commented out, or using Synaptic (10.04 & 10.10) or Ubuntu Software Center (11.04 onwards).

Now you need to update your package lists to ensure that you get the latest packages including those from the repositories that you just enabled:

```
$ sudo apt-get update
```

[56] http://www.zeroc.com

Java

From Ubuntu 11.10 and onwards, you can install OpenJDK 7 using:

```
$ sudo apt-get install openjdk-7-jdk
```

For earlier versions of Ubuntu, Oracle recently changed their distribution license (OSDL) which means that Debian and Ubuntu can no longer distribute Java in their package management systems. This means that you have to install the JDK separately as follows:

If you are on Debian then:

```
$ sudo apt-get install lsb_release
```

For both Debian and Ubuntu you can now:

```
$ sudo apt-get install git
$ git clone https://github.com/flexiondotorg/oab-java6.git
$ cd oab-java6/
$ sudo ./oab-java6.sh
```

The script will run and you should see some output indicating progress. When you get your prompt back:

```
$ sudo apt-get install sun-java6-jdk
```

You also need to ensure that the Sun/Oracle JDK is the active one as you end up with the OpenJDK also installed to satisfy dependencies along the way:

```
$ sudo update-alternatives --config java
```

and select the correct java from the displayed list.

OMERO dependencies

Now you are ready to install the rest of your prerequisite software packages:

```
$ sudo apt-get install unzip build-essential mencoder
$ sudo apt-get install python python-imaging python-numpy python-tables python-matplotlib
$ sudo apt-get install zeroc-ice33
$ sudo apt-get install postgresql
$ sudo apt-get install apache2 libapache2-mod-fastcgi
```

Note: On Ubuntu 11.04 and earlier, `python-tables` does not install. You need to install `liblzo` otherwise OMERO.tables will fail to start:

```
$ sudo apt-get install liblzo2-2
```

6.5.3 OMERO installation

Once the prerequisites are installed and configured, the OMERO.server can be set up. First, a home needs to be created for the server and this directory moved into. For example, to install OMERO locally into a directory called 'apps' in your home directory, use the following:

```
$ mkdir apps
$ cd apps
$ mkdir OMERO
$ cd OMERO
```

OMERO 4.4.12

Release versions of OMERO.server can downloaded from the OMERO downloads[57] page. Assuming that you downloaded a release version of OMERO.server, extract it from the zip archive:

```
$ unzip OMERO.server-4.4.12-ice33-byy.zip
```

Give your OMERO software install a nice local name to save some typing later, to reflect what you set `OMERO_PREFIX` to in the *Configuration* section, and to make it easy to manage the installation of newer versions of the server at a later date:

```
$ ln -s OMERO.server-4.4.12-ice33-byy  OMERO.server
```

Development server

If you want the development version of OMERO.server, you can clone the source code from the project's GitHub account to build locally:

```
$ git clone --recursive git://github.com/openmicroscopy/openmicroscopy
$ cd openmicroscopy && ./build.py
```

Note: If you have a GitHub account and you plan to develop code for OMERO, you should make a fork into your own account and then clone this fork to your local development machine, e.g.

```
$ git clone --recursive git://github.com/YOURNAMEHERE/openmicroscopy
$ cd openmicroscopy && ./build.py
```

See also:

Checking out the source code Developer documentation page on how to check out to source code

Build System Developer documentation page on how to build the OMERO.server

Alternatively, you can download a daily build[58] of the OMERO.server from our continuous integration server.

6.5.4 Configuration

Environment variables

> **Warning:** The `OMERO_HOME` environment variable is used internally by OMERO. Unless you really know what you are doing, it is strongly recommended not to set this variable.

Edit your `.bashrc` file, e.g. **$ vim ~/.bashrc** and add the following:

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun
export ICE_HOME=/usr/share/Ice-3.3.1
export POSTGRES_HOME=/usr/lib/postgresql/8.4
export OMERO_PREFIX=~/apps/OMERO/OMERO.server
```

[57] http://downloads.openmicroscopy.org/latest/omero4/
[58] http://ci.openmicroscopy.org/job/OMERO-trunk/lastSuccessfulBuild/artifact/

```
export PATH=$PATH:$JAVA_HOME/bin:$ICE_HOME:$POSTGRES_HOME/bin:$OMERO_PREFIX/bin
export PYTHONPATH=/usr/lib/pymodules/python2.6:$PYTHONPATH
export LD_LIBRARY_PATH=/usr/share/java:/usr/lib:$LD_LIBRARY_PATH
```

Note: You may wish to check PostgreSQL and Python versions by checking the directories themselves, since they may not correspond to those listed above. In particular check the version of Python that is installed. Newer versions of Ubuntu are installing Python 2.7 from APT by default.

Now you need to make those changes take effect by getting your shell to apply them using the **source** built-in command:

```
$ source ~/.bashrc
```

You can check that the new environment variables have taken by printing their values to the shell, e.g.:

```
$ echo $OMERO_PREFIX
/home/ome/apps/OMERO/OMERO.server
```

Database creation

Now you need to configure your prerequisites so that they are ready for OMERO to make use of. For the purposes of this walk-through you can use the following dummy data for the user account:

```
U: db_user
P: db_password
DB: omero_database
```

Note: For a live or public server install these values should be altered to reflect your security requirements. You should also consider locking down your server machine but that is outside the scope of this document.

Set up PostgreSQL:

```
$ sudo -u postgres createuser -P -D -R -S db_user
$ sudo -u postgres createdb -O db_user omero_database
$ sudo -u postgres createlang plpgsql omero_database
```

Check that a database called "omerodb" has been created:

```
$ psql -h localhost -U db_user -l
```

Update PostgreSQL host-based authentication to accept remote connections:

```
$ sudo sed '/127.0.0.1/s/md5/trust/' /etc/postgresql/8.4/main/pg_hba.conf \
    > pg_hba.conf && sudo mv pg_hba.conf /etc/postgresql/8.4/main/pg_hba.conf
```

Note: The backslash '\' in the sed command above is used merely to indicate a line-break and should not be included in the executed command

Restart PostgreSQL:

```
$ sudo /etc/init.d/postgresql restart
```

Use netstat to verify that there is something listening on port 5432, this should be your PostgreSQL server:

```
$ netstat -an | egrep '5432.*LISTEN'
```

which should display a line similar to the following:

```
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN
```

OMERO.server

Now you can configure OMERO.server so that it can connect to the PostgreSQL database:

```
$ omero config set omero.db.name 'omero_database'
$ omero config set omero.db.user 'db_user'
$ omero config set omero.db.pass 'db_password'
```

Note: If you altered any of these values earlier then you will need to change them to reflect your requirements

You can also check the values that have been set using:

```
$ omero config get
```

Create a home for your OMERO data. For example, to install the OMERO data locally into `~/apps/OMERO/OMERO.data`, use the following command:

```
$ mkdir ~/apps/OMERO/OMERO.data
```

Configure OMERO to find the data location:

```
$ omero config set omero.data.dir ~/apps/OMERO/OMERO.data
```

You can now configure the empty PostgreSQL database using Omero's db script. You can accept the defaults for the first few values and enter a suitable password as required when prompted, e.g. "root_password:

```
$ omero db script
```

The output of this should be a file named, e.g. OMERO4.4__0.sql file in your current directory. You can now tell PostgreSQL to configure your new database

```
$ psql -h localhost -U db_user omero_database < OMERO4.4__0.sql
```

At this point you should see a whole load of output from PostgreSQL as it installs the new OMERO database.

If all has gone well, you should now be able to start OMERO.server using the following command:

```
$ omero admin start
```

You should now be able to connect to your OMERO.server using an OMERO client such as OMERO.insight and the following credentials:

```
U: root
P: root_password
```

OMERO.web

To connect with the webclient or webadmin using the included Django development server:

```
$ omero config set omero.web.application_server development
$ omero config set omero.web.debug True
```

You should be able to start the Web server with:

```
$ omero web start
Starting django development webserver...
Validating models...
0 errors found

Django version 1.1.1, using settings 'omeroweb.settings'
Development server is running at http://0.0.0.0:4080/
Quit the server with CONTROL-C.
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

6.6 OMERO.web deployment

OMERO.web is the web application component of the OMERO platform which allows for the management, visualization (in a fully multi-dimensional image viewer) and annotation of images from a web browser. It also includes OMERO.webadmin for managing users and groups.

OMERO.web is an integral part of the OMERO platform and can be deployed with:

- FastCGI using a FastCGI capable web server such as Apache[59] (with mod_fastcgi[60] enabled), nginx[61] or lighttpd[62] (since OMERO 4.2.1)
- The built-in Django lightweight development server (for **testing** only)

You can find more information about FastCGI and where to get modules or packages for your distribution on the FastCGI website[63].

If you need help configuring your firewall rules, see the *Server security and firewalls* page.

6.6.1 Prerequisites

- OMERO and its prerequisites (see *OMERO.server installation*).
- Python version from 2.4 to 2.7 (due to backwards incompatibilities in Python 3.0, Django 1.3 does not work with Python 3.0; for more information see the Django Installation page[64]).
  - Python Imaging Library[65] should be available for your distribution
  - Matplotlib[66] should be available for your distribution.

[59] http://httpd.apache.org/
[60] http://www.fastcgi.com/drupal/
[61] http://nginx.org/
[62] http://www.lighttpd.net/
[63] http://www.fastcgi.com/drupal/node/3
[64] https://docs.djangoproject.com/en/1.3/intro/install/
[65] http://www.pythonware.com/products/pil/
[66] http://matplotlib.org/

- A FastCGI capable web server

6.6.2 Configuring OMERO from the command line

Configuration options can be set using the `omero config set` command:

```
$ bin/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:

```
$ bin/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:

```
$ bin/omero config get
```

which should return values without quotation marks.

A final useful option of `omero config edit` is:

```
$ bin/omero config edit
```

which will allow for editing the configuration in a system-default text editor.

6.6.3 Quick Start

Using FastCGI (Unix/Linux)

Once you have installed a FastCGI capable web server, configuration of OMERO.web is quite straightforward.

- Choose between FastCGI TCP (recommended) or FastCGI (advanced):

```
$ bin/omero config set omero.web.application_server "fastcgi" / "fastcgi-tcp"
```

- Place a stanza in your web server configuration file. The location of the file will depend on your system, please refer to your web server's manual. Apache and nginx configurations can be automatically generated for you by OMERO.web, see *Apache configuration* or *Nginx configuration*.
- Start the Django FastCGI workers:

```
$ bin/omero web start
....
Copying '/Users/omero/Desktop/omero/lib/python/omeroweb/webstart/static/webstart/img/icon-omero-we

735 static files copied to '/Users/omero/Desktop/omero/lib/python/omeroweb/static'.
Starting OMERO.web... [OK]
```

Note: The Django FastCGI workers are managed **separately** from other OMERO.server processes. You can check their status or stop them using the following commands:

```
$ bin/omero web status
OMERO.web status... [RUNNING] (PID 59217)
$ bin/omero web stop
Stopping OMERO.web... [OK]
Django FastCGI workers (PID 59217) killed.
```

Apache configuration

To create a site configuration file for inclusion in the main Apache configuration redirect the output of the following command into a file:

```
$ bin/omero web config apache
```

```
...
###
### Stanza for OMERO.web created 2012-07-12 16:44:16.112099
###
FastCGIExternalServer "/usr/local/dev/openmicroscopy/dist/var/omero.fcgi" -host 0.0.0.0:4080

<Directory "/usr/local/dev/openmicroscopy/dist/var">
    Options -Indexes FollowSymLinks
    Order allow,deny
    Allow from all
</Directory>

<Directory "/usr/local/dev/openmicroscopy/dist/lib/python/omeroweb/static">
    Options -Indexes FollowSymLinks
    Order allow,deny
    Allow from all
</Directory>

Alias /static /usr/local/dev/openmicroscopy/dist/lib/python/omeroweb/static
Alias /omero "/usr/local/dev/openmicroscopy/dist/var/omero.fcgi/"
```

Note: The default configuration file installed with *mod_fastcgi* may be incompatible with OMERO. In particular, the *FastCGIWrapper* option conflicts with *FastCGIExternalServer* required by OMERO and must be removed or commented out.

Nginx configuration

To create a configuration file for a standalone instance of nginx redirect the output of the following command into a file:

```
$ bin/omero web config nginx
```

```
#
# nginx userland template
# this configuration is designed for running nginx as the omero user or similar
# nginx -c etc/nginx.conf
# for inclusion in a system-wide nginx configuration see omero web config nginx --system
#
pid /usr/local/dev/openmicroscopy/dist/var/pid.nginx;
error_log /usr/local/dev/openmicroscopy/dist/var/log/nginx_error.log;
worker_processes  5;
working_directory /usr/local/dev/openmicroscopy/dist/var;
```

```
events {
    worker_connections  1024;
}


http {
    access_log    /usr/local/dev/openmicroscopy/dist/var/log/nginx_access.log;
    include    /usr/local/dev/openmicroscopy/dist/etc/mime.types;
    default_type  application/octet-stream;
    client_body_temp_path /usr/local/dev/openmicroscopy/dist/var/nginx_tmp;

    keepalive_timeout  65;

    server {
        listen       8080;
        server_name  _;
        fastcgi_temp_path /usr/local/dev/openmicroscopy/dist/var/nginx_tmp;
        proxy_temp_path /usr/local/dev/openmicroscopy/dist/var/nginx_tmp;

         # weblitz django apps serve static content from here
        location /static {
            alias /usr/local/dev/openmicroscopy/dist/lib/python/omeroweb/static;
        }

        location / {
            if (-f /usr/local/dev/openmicroscopy/dist/var/maintenance.html) {
               error_page 503 /maintenance.html;
               return 503;
            }

            fastcgi_pass 0.0.0.0:4080;

            fastcgi_param PATH_INFO $fastcgi_script_name;


            fastcgi_param REQUEST_METHOD $request_method;
            fastcgi_param QUERY_STRING $query_string;
            fastcgi_param CONTENT_TYPE $content_type;
            fastcgi_param CONTENT_LENGTH $content_length;
            fastcgi_param SERVER_NAME $server_name;
            fastcgi_param SERVER_PROTOCOL $server_protocol;
            fastcgi_param SERVER_PORT $server_port;
            fastcgi_pass_header Authorization;
            fastcgi_intercept_errors on;
            fastcgi_read_timeout 300;
            # Uncomment if nginx SSL module is enabled or you are using nginx 1.1.11 or later
            # -- See: #10273, http://nginx.org/en/CHANGES
            # fastcgi_param HTTPS $https;
        }

        location /maintenance.html {
            root /usr/local/dev/openmicroscopy/dist/var;
        }

    }

}
```

It is also possible to generate a site file for inclusion as part of a system-wide nginx instance:

```
$ bin/omero web config nginx --system
```

See also:

Custom OMERO.web location.

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, turn Debugging on and start the server up:

```
$ bin/omero config set omero.web.application_server development
$ bin/omero config set omero.web.debug True
$ bin/omero web start
Copying '/Users/omero/Desktop/omero/lib/python/omeroweb/feedback/static/feedback/css/layout.css'
.....
Copying '/Users/omero/Desktop/omero/lib/python/omeroweb/webstart/static/webstart/img/icon-omero-web.png

735 static files copied to '/Users/omero/Desktop/omero/lib/python/omeroweb/static'.
Starting OMERO.web... Validating models...

0 errors found
Django version 1.3.1, using settings 'omeroweb.settings'
Development server is running at http://0.0.0.0:4080/
Quit the server with CONTROL-C.
```

6.6.4 Logging in to OMERO.web

Once you have deployed and started the server, you can use your browser to access OMERO.webadmin or the OMERO.webclient:

- **http://your_host/omero** OR, for development server: **http://localhost:4080**

Figure 6.2: OMERO.webadmin login

Note: This starts the server in the foreground. It is your responsibility to place it in the background, if required, and manage its shutdown.

6.6.5 Customizing your OMERO.web installation

Note: For clarity, some edge-case/in-development options may not be documented below. For the full list see:

```
$ bin/omero web -h
```

OR look in lib/python/omeroweb/settings.py

- A list of servers the Web client can connect to. Default: `[["localhost", 4064, "omero"]]`.

```
$ bin/omero config set omero.web.server_list '[["prod.example.com", 4064, "prod"], ["dev.example.co
```

- Email server and notification:
  - (**REQUIRED**) From : address to be used when sending e-mail. Default: `root@localhost`

```
$ bin/omero config set omero.web.server_email "webmaster@example.com"
```

  - (**REQUIRED**) Mail server hostname. Default: `localhost`.

```
$ bin/omero config set omero.web.email_host "email.example.com"
```

  - Mail server login username. Default: '' (Empty string).

```
$ bin/omero config set omero.web.email_host_user "username"
```

  - Mail server login password. Default: '' (Empty string).

```
$ bin/omero config set omero.web.email_host_password "password"
```

  - Mail server port. Default: `25`.

```
$ bin/omero config set omero.web.email_host_port "2255"
```

  - Use TLS when sending e-mail. Default: `False`.

```
$ bin/omero config set omero.web.email_use_tls "True"
```

  - Subject prefix for outgoing e-mail. Default: `"[Django] "`.

```
$ bin/omero config set omero.web.email_subject_prefix "Subject prefix for outgoing e-mail"
```

- Controlling displayed scripts:
  - Since OMERO 4.3.2, OMERO.web has the ability to dynamically display scripts in the script runner menu just like OMERO.insight. Some scripts were not suitable for display initially and are excluded from the menu. You may wish to control which scripts your users can see in OMERO.web using this configuration option. Default: `'["/omero/figure_scripts/Movie_Figure.py", "/omero/figure_scripts/Split_View_Figure.py", "/omero/figure_scripts/Thumbnail_Figure.p "/omero/figure_scripts/ROI_Split_Figure.py", "/omero/export_scripts/Make_Movie.py"]'`

```
$ bin/omero config set omero.web.scripts_to_ignore '[]'
$ bin/omero config set omero.web.scripts_to_ignore '["/omero/my_scripts/really_buggy.py", … ]'
```

- Enabling a public user (automatically log in a public user, OMERO 4.4.0 onwards):

  - First, create a public user. You can use any username and password you wish. If you do not want this user to be able to modify any of the data they see, you should put this user in a Read-Only group and the public data should be owned by another member(s) of this group.
  - Enable the OMERO.web public user functionality, which is disabled (False) by default.

```
$ bin/omero config set omero.web.public.enabled True
```

  - Set a URL filter for which the OMERO.web public user is allowed to navigate. Default: `^/(?!webadmin)` (Python regular expression[67]). You probably do not want the whole webclient UI to be publicly visible (although you could do this). The idea is that you can create the public pages yourself (see *OMERO.web framework*) since we do not provide public pages. E.g. to allow only URLs that start with '/my_web_public' you would use:

```
$ bin/omero config set omero.web.public.url_filter '/my_web_public'
```

    Exotic matching techniques can be used but more explicit regular expressions are needed when attempting to filter based on a base URL:

```
'webtest' matches '/webtest' but also '/webclient/webtest'
'dataset' matches '/webtest/dataset' and also '/webclient/dataset'
'/webtest' matches '/webtest...' but also '/webclient/webtest'
'^/webtest' matches '/webtest...' but not '/webclient/webtest'
```

    If you need more examples of how to configure public url filters, see the *Public data in OMERO.web* page.
  - Server to authenticate against. Default: `1` (the first server in `omero.web.server_list`)

```
$ bin/omero config set omero.web.public.server_id 1
```

  - Username to use during authentication. Default: `Not set.` (required if `omero.web.public.enabled=True`):

```
$ bin/omero config set omero.web.public.user '__public__'
```

  - Password to use during authentication. Default: `Not set.` (required if `omero.web.public.enabled=True`):

```
$ bin/omero config set omero.web.public.password 'secret'
```

- Administrator e-mail notification:
  - Admins list of people who get code error notifications. When debug mode is off and a view raises an exception, Django will e-mail these people with the full exception information. Default: `[]` (Empty list).

```
$ bin/omero config set omero.web.admins '[["Dave", "dave@example.com"], ["Bob", "bob@example.co
```

- Ping interval:
  - Since OMERO 4.4.0, OMERO.web now pings the server to keep your session alive when you are logged in and have an active browser window. The duration between these pings can be configured. Default: `60000.` (every 60 seconds)

```
$ bin/omero config set omero.web.ping_interval 12000
```

- Debug mode:

[67] http://docs.python.org/2/library/re.html

  - A boolean that turns on/off debug mode. Default: `False`.

```
$ bin/omero config set omero.web.debug "True"
```

- Custom OMERO.web location:
  - By default OMERO.web expects to be run from the root URL of the web server. It can be configured to run from a sub-directory, for example *http://example.org/testing/*, as follows:

```
$ bin/omero config set omero.web.force_script_name '/testing'
```

  - Update your web server configuration. For example, in the nginx configuration replace

```
fastcgi_param PATH_INFO $fastcgi_script_name;
```

    with

```
fastcgi_split_path_info ^(/testing)(.*)$;
fastcgi_param PATH_INFO $fastcgi_path_info;
fastcgi_param SCRIPT_INFO $fastcgi_script_name;
```

  - If *omero.web.force_script_name* is set then

```
$ bin/omero web config nginx
```

    will automatically generate the correct configuration.
- Configuring additional web apps:
  - The OMERO.web framework allows you to add additional Django apps. For an example with installation instructions, see webmobile[68]
  - Download or clone from the git repository into the /omeroweb/ directory, then run

```
$ bin/omero config set omero.web.apps '["<app name>"]'
```

[68] https://github.com/will-moore/webmobile/

CHAPTER

SEVEN

BASIC WINDOWS SERVER INSTALLATION

This chapter contains the instructions to install OMERO.server on Windows platforms.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

7.1 OMERO.server installation

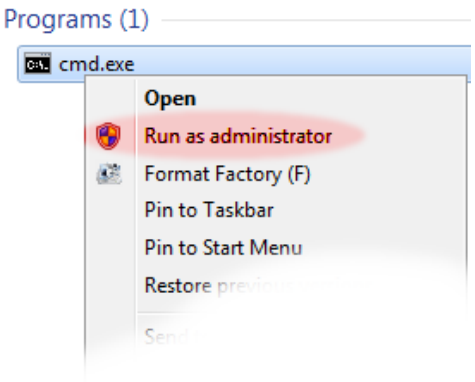
See also:

OMERO.server upgrade Instructions for **upgrading** your OMERO.server installation.

7.1.1 Limitations

Installation **will require an "Administrator" level account** for which you know the password. If you are unsure of what it means to have an "Administrator" 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?*.

- Unless you are clear on the differences, **you should also open all consoles as an administrator to prevent file permission issues.**



- Installation on Windows XP is not explicitly supported, especially for OMERO.web. Significant testing has taken place on Windows Server 2008 and we recommend this version.
- OMERO does not currently support Ice 3.5 or Python 3.
- *OMERO.movie* is not supported on Windows at present.
- Spaces in installation path names are not currently supported - **do not use spaces in your folder names**:

  Note: The default user paths on Windows usually contain spaces so you will need to ensure the path has no spaces, `C:\OMERO.server` for example.

- A reboot is required after installing the prerequisites.

7.1.2 Prerequisites

Note: The installation of these prerequisite applications is largely outside the scope of this document.

Do not mix 32bit (x86) and 64bit (amd64) packages - install either all 32bit or all 64bit. Check your JRE as well.

The following are necessary:

Java SE Development Kit (JDK)

Java SE Downloads are available from http://www.oracle.com/technetwork/java/javase/downloads/index.html. JDK 6 and above are supported.

Ice (3.3.x or 3.4.x)

Note: OMERO 4.3 and earlier support only Ice 3.3. You will need to download these from ZeroC's previous versions section[1]. OMERO 4.4 adds support for Ice 3.4 while keeping support for Ice 3.3. For OMERO.server you will need to pick the appropriate downloads for the version of Ice you have installed locally. The downloads for Ice 3.4 have "ice34" in the zip name.

OMERO does not currently support Ice 3.5 for Windows - if you have installed Ice 3.5, uninstall it, install 3.4.x, update ENV path and reboot. If you need to use Ice 3.5 for other purposes, you probably just need to add the path for 3.4.x to the ENV before Ice 3.5.

Windows installers of Ice can be found on the ZeroC download page[2] and will be called something like `Ice-3.4.2.msi` (for Ice 3.4.2). If you plan to develop for C++, be sure to read the instructions on the *OMERO C++ language bindings* page.

Users have reported that under certain conditions installing Ice in the default location (`C:\Program Files`) might break the Windows `PATH` (due to special characters present in the file name). In cases where `omero admin diagnostics` throws any errors related to unexpected paths with Ice, please consider installing Ice in the root of the partition (e.g. `C:\Ice`).

Python 2 (2.4 or higher)

Ice 3.4.x requires Python 2.6.x. You must download the 32-bit version from python.org[3]. As this is the "vanilla" python distribution (no extra libraries), you will need to install further dependencies, making sure to download the correct version (release number, 32/64-bit) for your Python distribution.

OMERO does not currently support Python 3.

PyWin32

Using **Python for Windows extensions** is recommended. The installer is available from the PyWin download page[4].

The version you need is: `pywin32-XXX.win32-pyA.B.exe` (or the 64bit version) where XXX should be the latest release number and A and B stand for the Python version e.g. `pywin32-218.win32-py2.6.exe`.

You can read the readme.txt[5] to be sure of which file to download.

Additional libraries

The following are optional depending on what services you require:

[1] http://zeroc.com/previous.html
[2] http://zeroc.com/download_3_4_2.html
[3] http://www.python.org/download/releases/2.6.6/
[4] http://sourceforge.net/projects/pywin32/files/pywin32/
[5] http://sourceforge.net/projects/pywin32/files/pywin32/Build%20218/README.txt/download

Package	Functionality	Downloads
Python Imaging Library	OMERO.web and Figure Export	PIL page[14]
Matplotlib	OMERO.web	Matplotlib page[15]
NumPy (1.2.0 or higher) [16]	Scripting	Numpy/Scipy page[17]
PyTables (2.1.0 or higher)	*OMERO.tables*	PyTables page[18]
scipy.ndimage	Volume Viewer[19] [20]	Numpy/Scipy page[21]

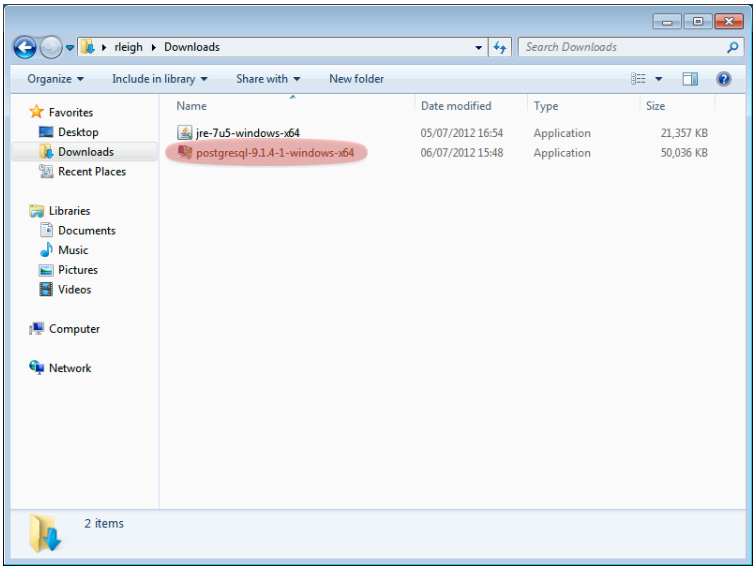
PostgreSQL (8.4 or higher)

PostgreSQL has to be installed and configured with PL/pgSQL and to accept TCP connections. PostgreSQL 8.3 and earlier are not supported.

The *One click installer* can be found on the PostgreSQL Windows download page[22]. You will need the postgres user's password later in the install.

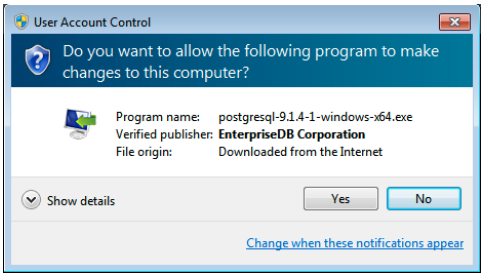
- You must install PostgreSQL as a service if you want to follow this document; other PostgreSQL installation environments are supported but are outside the scope of this document.

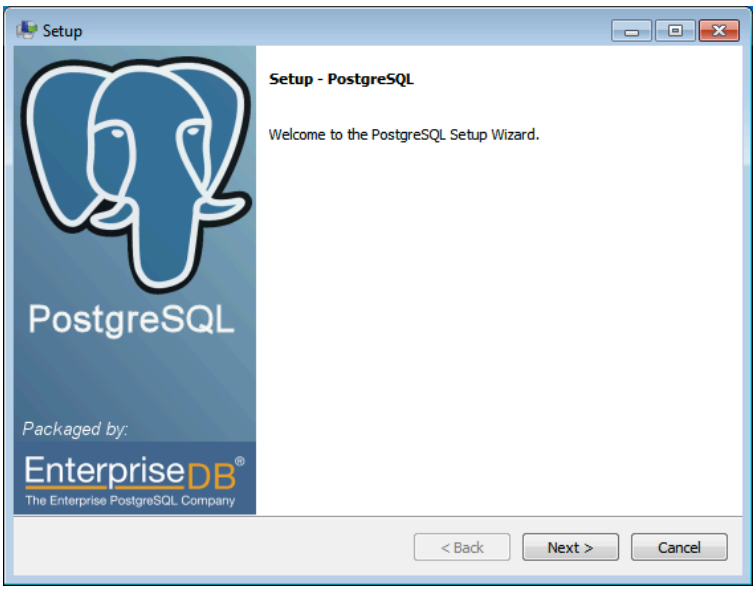
1. Run the downloaded installer:



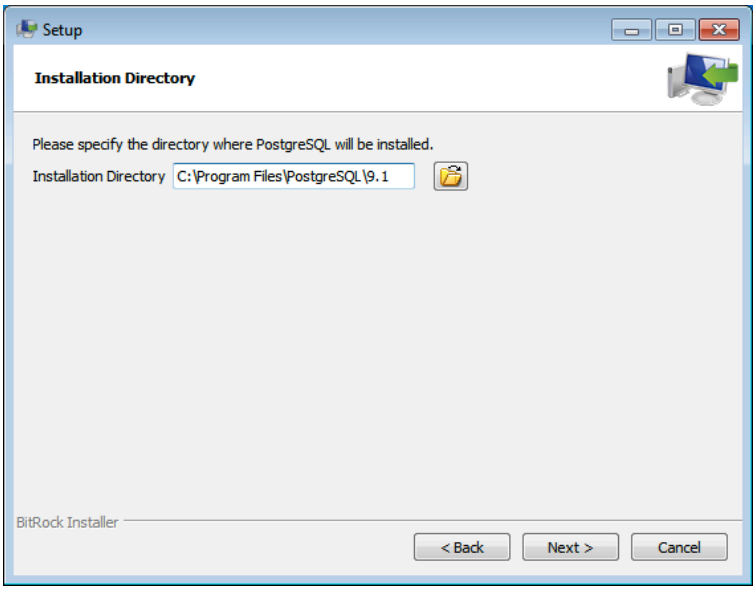
2. You may be prompted for permission to continue with a "user account control" dialog. Click *yes* to continue.
3. The installer will now start.
4. Choose the installation directory. The default is fine.
5. Choose the data directory. The default is fine, but if you want to keep the data in a specific location, you may choose an alternative location here.

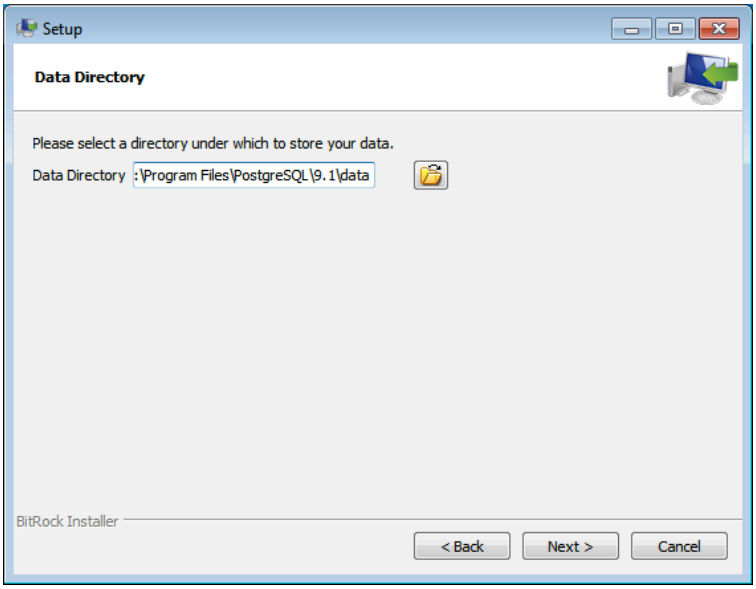
[6] http://www.pythonware.com/products/pil/
[7] http://matplotlib.org/
[8] May already have been installed as a dependency of Matplot Lib.
[9] http://www.scipy.org/Download
[10] http://www.pytables.org/moin/Downloads
[11] http://www.openmicroscopy.org/site/products/omero/volume-viewer-in-omero.web
[12] Allows larger volumes to be viewed in the Volume Viewer (http://www.openmicroscopy.org/site/products/omero/volume-viewer-in-omero.web).
[13] http://www.scipy.org/Download
[14] http://www.pythonware.com/products/pil/
[15] http://matplotlib.org/
[16] May already have been installed as a dependency of Matplot Lib.
[17] http://www.scipy.org/Download
[18] http://www.pytables.org/moin/Downloads
[19] http://www.openmicroscopy.org/site/products/omero/volume-viewer-in-omero.web
[20] Allows larger volumes to be viewed in the Volume Viewer (http://www.openmicroscopy.org/site/products/omero/volume-viewer-in-omero.web).
[21] http://www.scipy.org/Download
[22] http://www.postgresql.org/download/windows

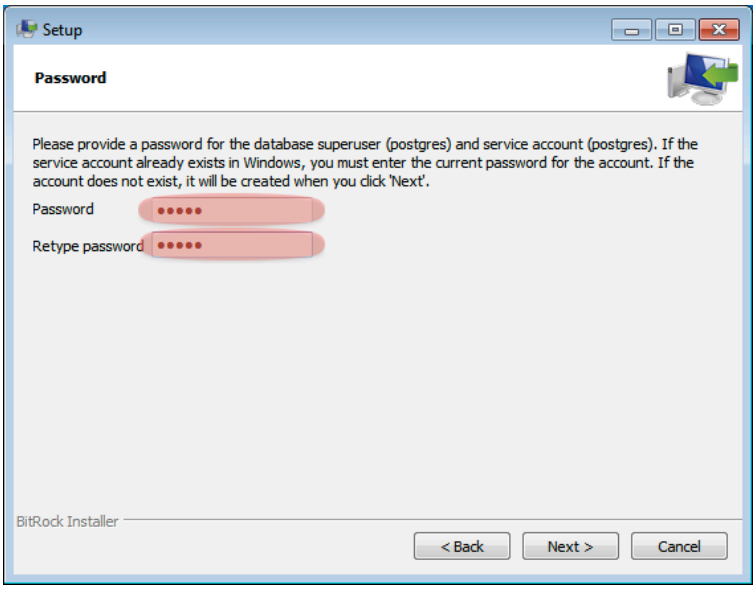


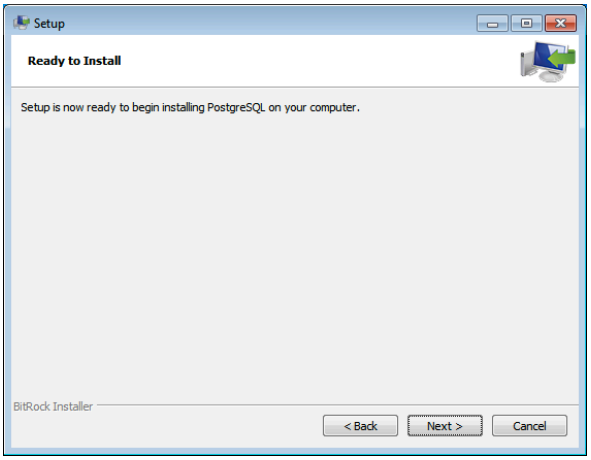


6. Enter a password for the special "postgres" system account. OMERO does not use this account, but you will need to remember the password for creating the database, below.
7. Enter the port number for PostgreSQL to listen on for incoming connections. The default, 5432, is fine and should not be changed.
8. Select the locale. The default here is fine.
9. PostgreSQL will now be installed and started.



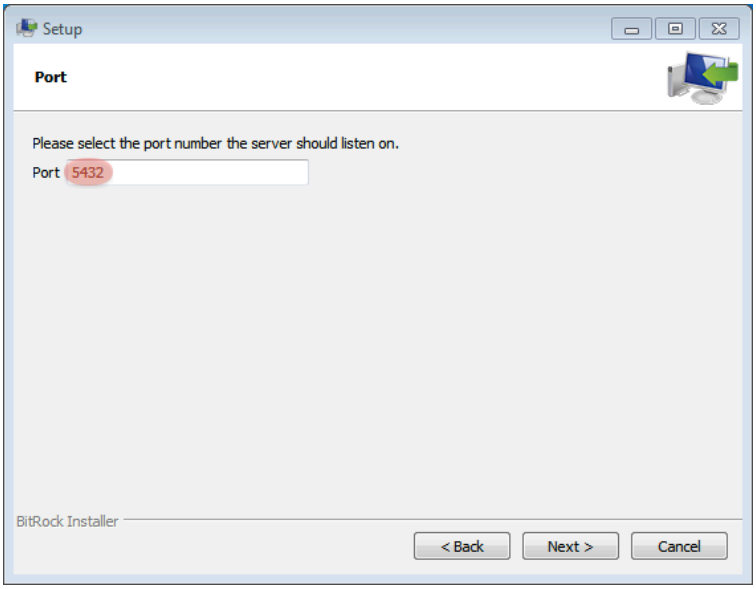


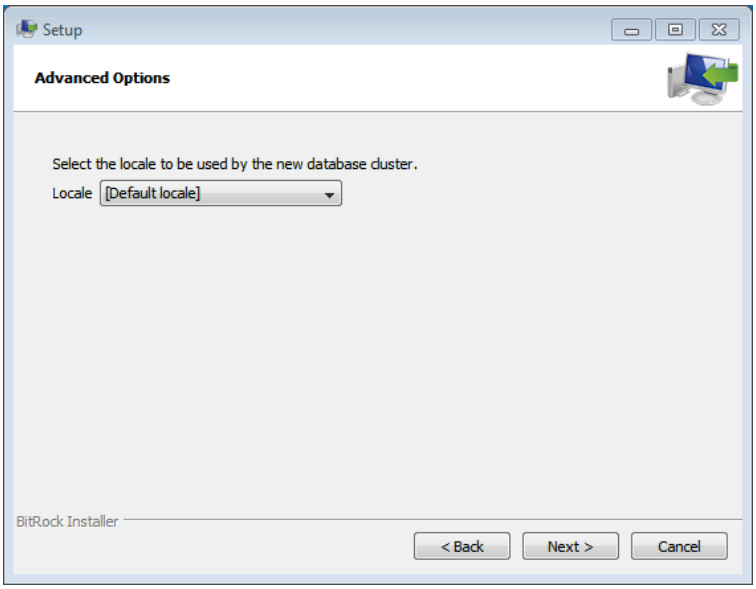




7.1.3 Environment variables

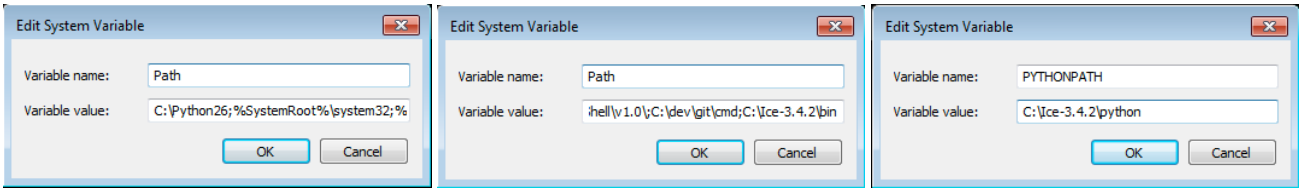
For the prerequisite software to run properly, your PATH and PYTHONPATH environment variables must be configured. If not correctly configured or if you installed any of the prerequisites manually to `C:\ice` or a similar location, you will need to set the values yourself.





Update your Windows environment variables: (REQUIRES RESTART!)

1. Locate the *System* control panel page on the Start Menu under *Settings* → *Control Panel*, open it and navigate to the *Advanced* tab (on Windows Vista the dialog will be visible after clicking the *Change settings* link on the *System* control panel page):
2. Open the *Environment Variables* dialog by clicking on the *Environment Variables...* button of the above dialog:
3. Edit the existing *System* environment variable *Path* and add a new variable pointing to the Ice installation `bin` directory. At the front of the *Path* variable also add a new string pointing to the Python installation directory (e.g. `C:\Python26`). Then add a brand new *System* environment variable called *PYTHONPATH* pointing to the Ice installation `python` location:



When setting the ENV variables, make sure you write in the correct paths. You must have entries for:

- python (the first PATH entry, e.g. "C:\Python26;%Sys…)
- ice/bin (the last PATH entry, e.g. "…;C:\Ice-3.4.2\;")
- PYTHONPATH pointing to the python folder in the ICE installation (e.g. "C:\Ice-3.4.2\python;")

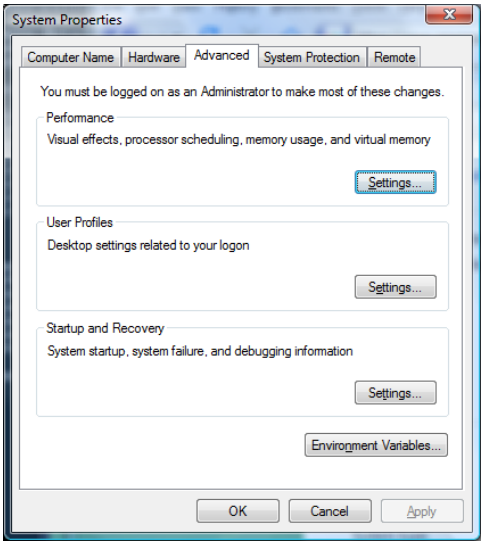


Figure 7.1: Advanced System Properties

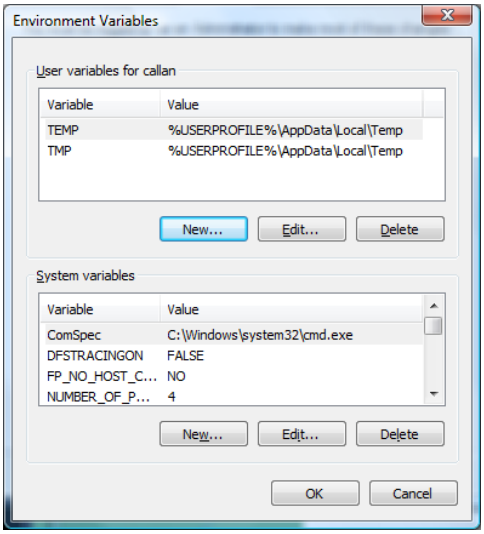


Figure 7.2: Environment Variables

Warning: Remember that the Windows path separator is the semicolon ; and must be appended after every entry. Make sure the first inserted ENV PATH entry (the python path) finishes with a semicolon (eg. "C:\Python26;%SystemRoot%…") otherwise you could corrupt other system applications.

4. **Restart your computer**. For environment changes to take effect in background services, a restart is unfortunately necessary. See http://support.microsoft.com/kb/821761 for more information.

7.1.4 Creating a database for OMERO

Probably the most important step towards having a working server system is having a properly configured database.

- Create a non-superuser database user (make sure to note down the name and password) using pgAdmin III. You can find pgAdmin III on the Start Menu under *Programs* → *PostgreSQL 9.1* → *pgAdmin III*:
  1. Double-click on the *PostgreSQL 9.1* database (or right- click and choose *Connect*) and provide your *postgres* user login password set during the installation, above.
  2. Right-click on *Login Roles* and select *New Login Role...*
  3. Create a new role and record the name and password. You will need to configure OMERO to use your username and password by setting the `omero.db.name` and `omero.db.pass` properties.

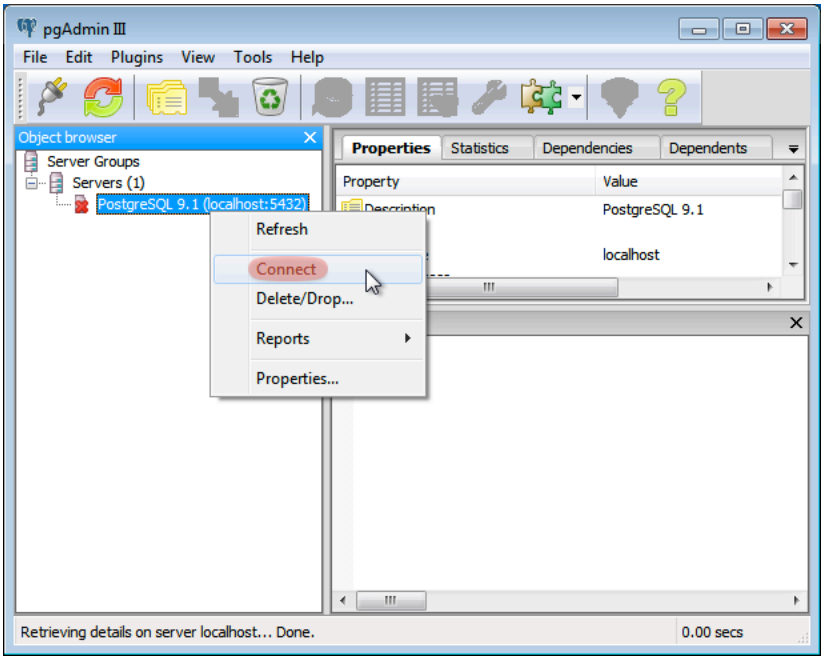


Figure 7.3: Connect to the database server

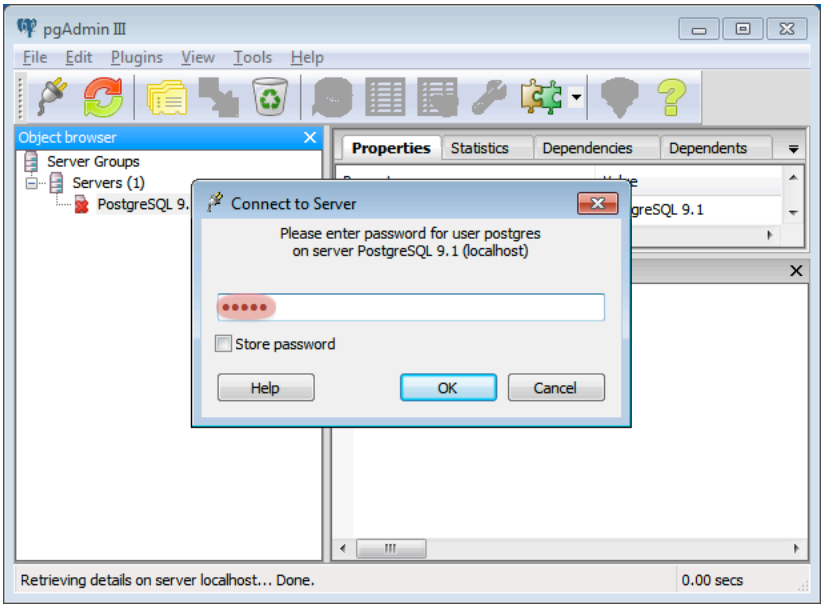


Figure 7.4: Enter password

Warning: For illustrative purposes, the default name and password for the role are `db_user` and `db_password` respectively. However, you should not use these default values for your installation but use your own choice of username and password instead.

- Create an `omero_database` database:
  1. Right-click on *Databases* and select *New Database* ...
  2. Create a new database with the *Name* `omero_database` and *Owner* `db_user` (this may take a few moments)
- Confirm PL/pgSQL language support in your newly created database
  1. First, go to *File* → *Options* select the *Browser* tab and activate the *Languages* option:
  2. Navigate back to your database, expand the database's tree view and finally expand the now available *Languages* item:

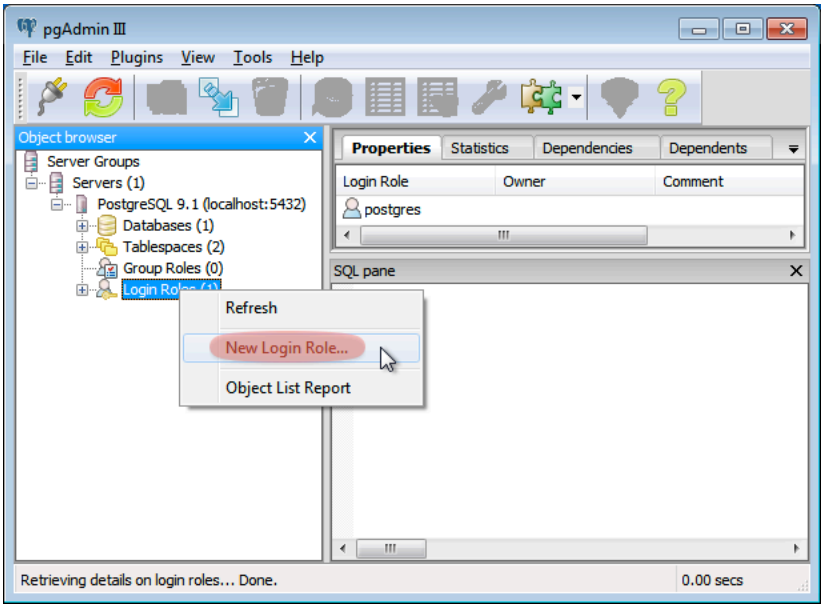


Figure 7.5: New login role

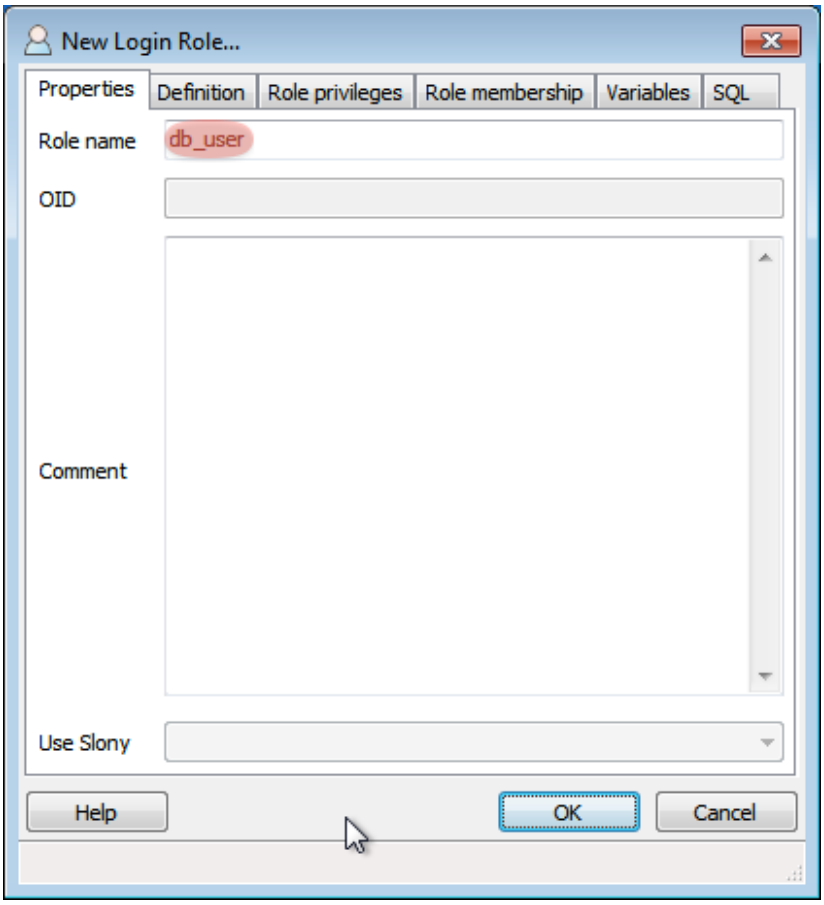


Figure 7.6: Setting name of new login role

3. If the `plpgsql` language is missing, right-click on the *Extensions* item and select the *New extension...* option in the menu. Finally, add the `plpgsql` extension, accepting all defaults. This will add both the extension and the language. In older PostgreSQL versions without extensions, right-click on the *Languages* item and select the *New language...* option in the menu. Finally, add the `plpgsql` language, accepting all defaults.

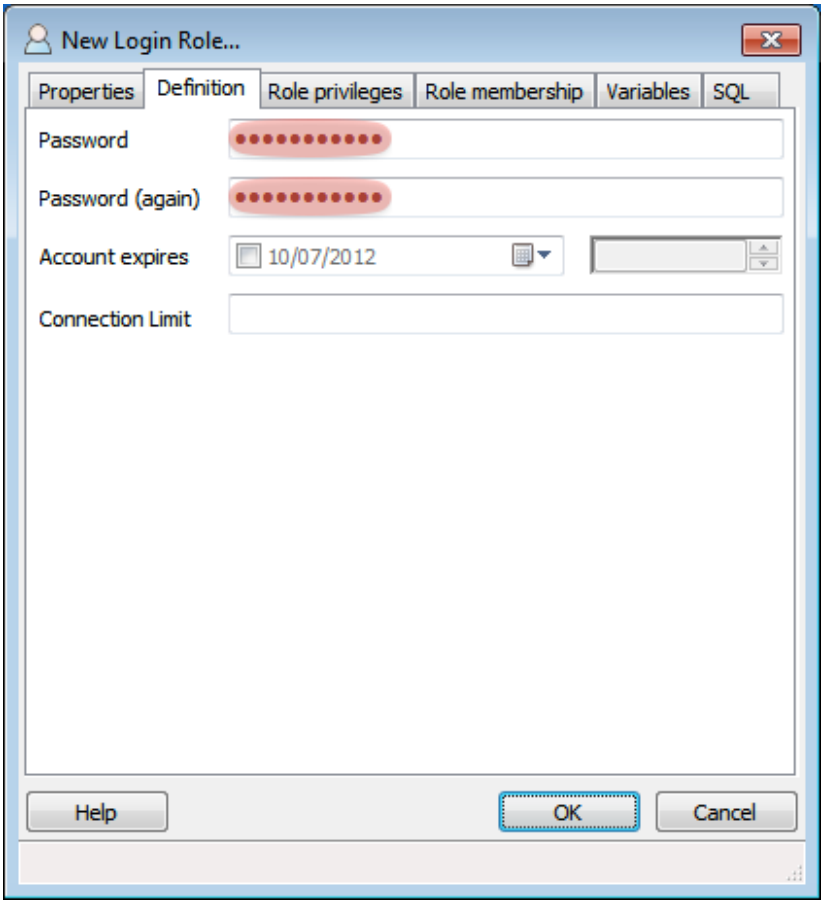


Figure 7.7: Setting password of new login role

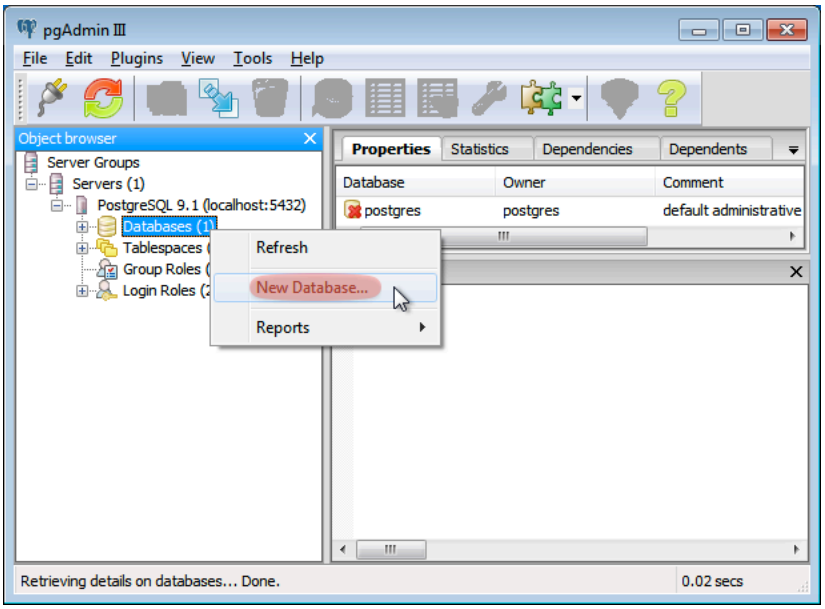


Figure 7.8: New database

7.1.5 Location for your OMERO binary repository

See also:

OMERO.server binary repository

- Create a directory for the OMERO binary data repository (`C:\OMERO` is the default location and should be used unless you explicitly have a reason not to and know what you are doing).

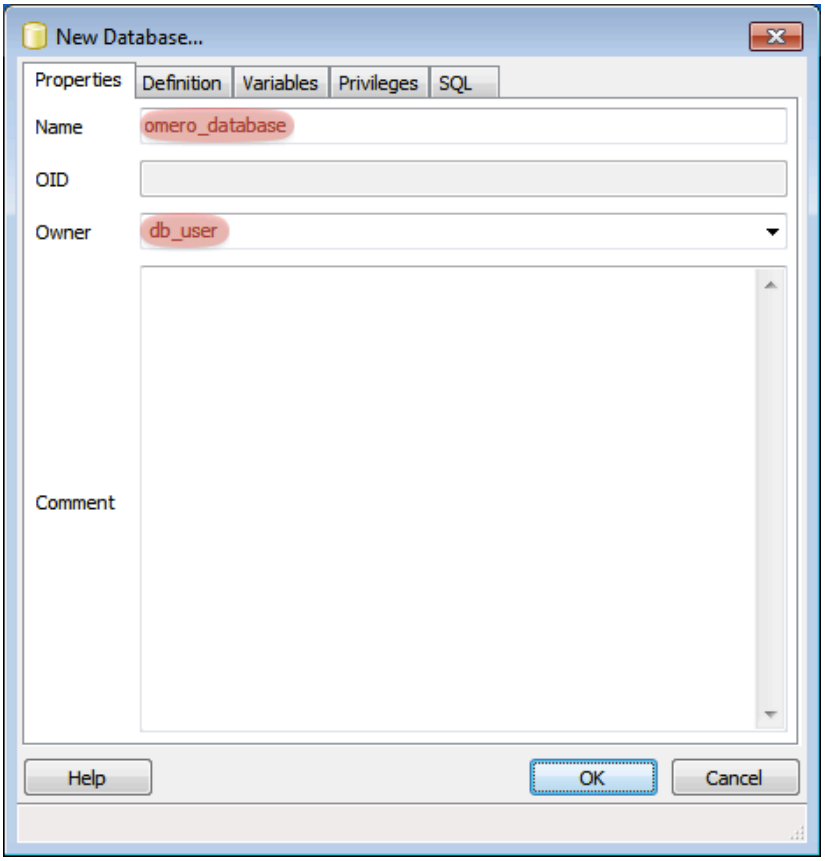


Figure 7.9: New database name

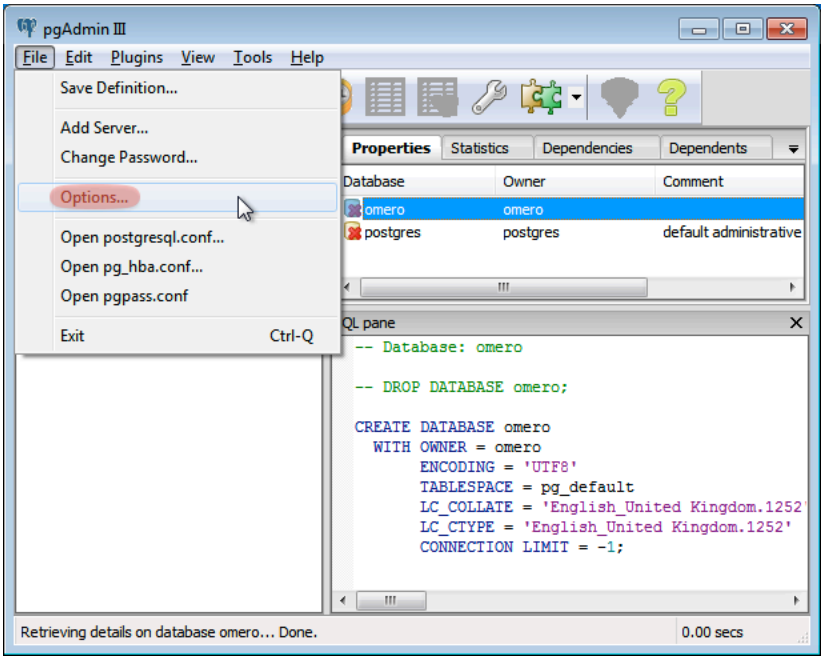


Figure 7.10: Options menu

- This is *not* where you want the OMERO application to be installed, it is a *separate* directory that OMERO.server will use to store binary data:

```
C:\mkdir OMERO
```

- Change the ownership of the directory. `C:\OMERO` *must* either be owned by the user starting the server or that user *must*

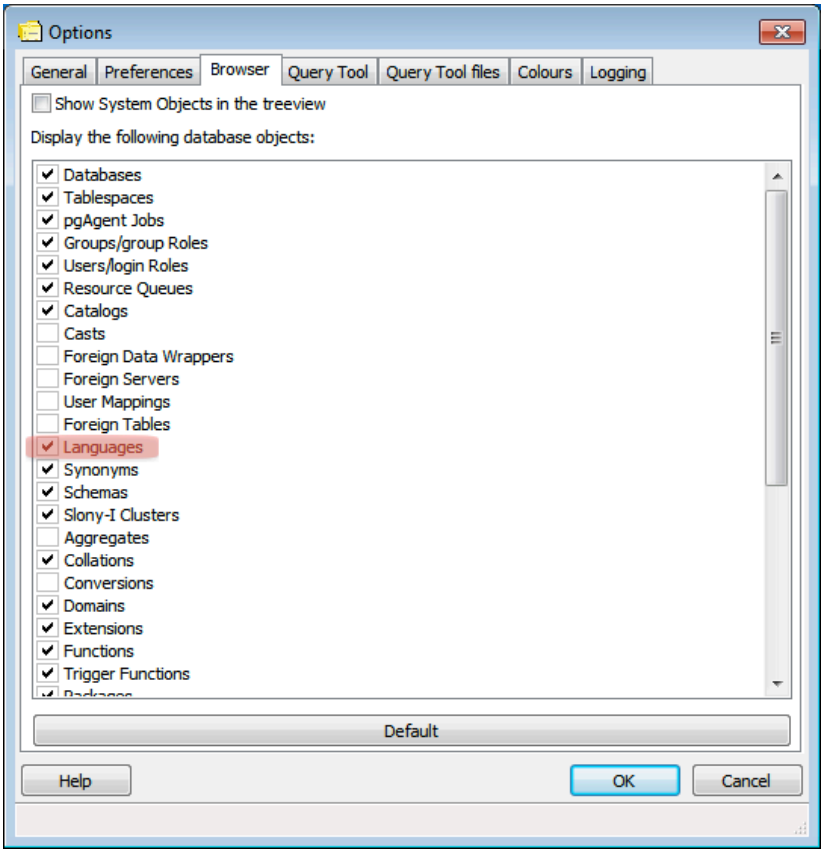


Figure 7.11: Enable display of installed languages

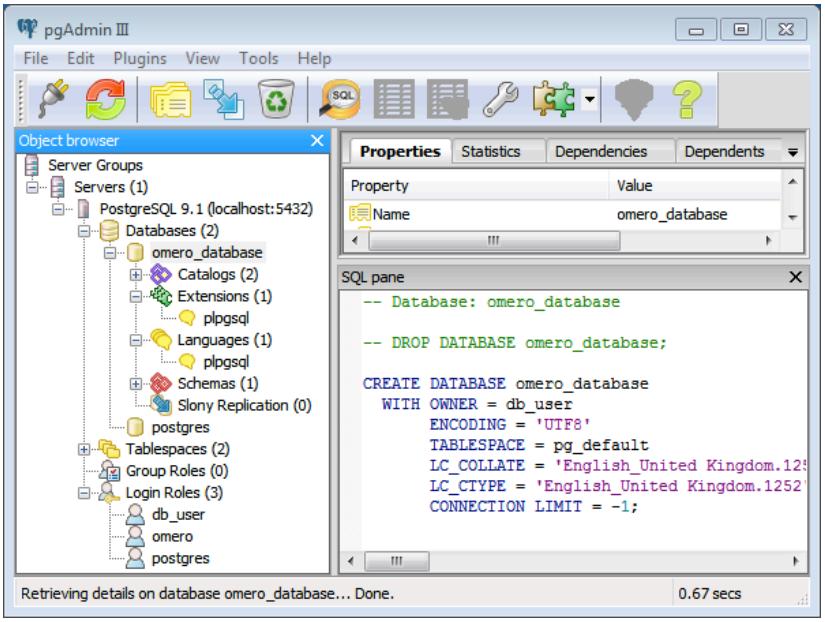


Figure 7.12: View installed languages

have permission to write to the directory. Please see *OMERO.server binary repository* for more details.

When performing some operations the clients make use of temporary file storage and log directories. By default these files are stored below the user's home directory (on Windows `C:\Users\<username>`) in `omero\tmp`, `omero\log` and `omero\sessions`.

Note: If your home directory is stored on a network (NFS mounted or similar), then file read and write operations occur over the network. This can slow access down. Installing OMERO on a network mapped drive is strongly discouraged.

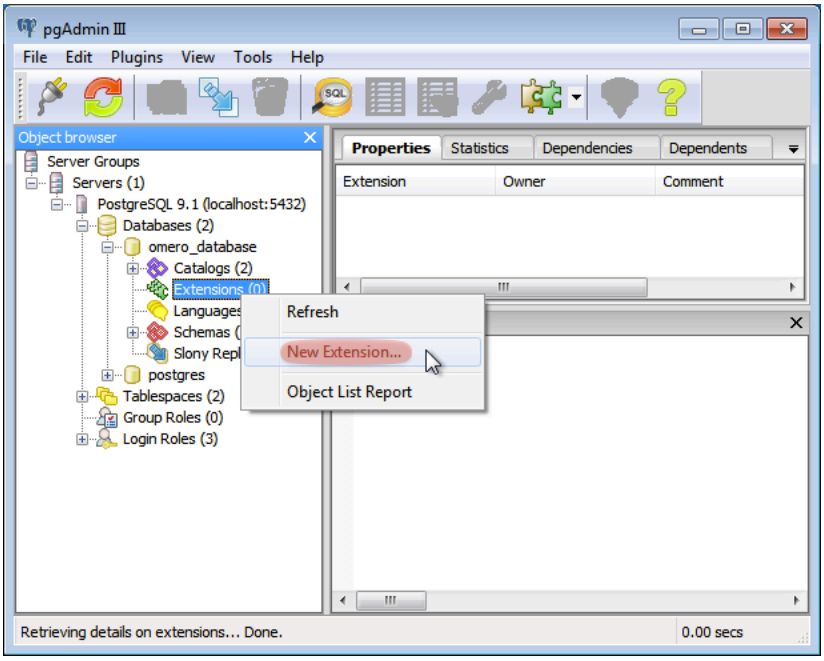


Figure 7.13: Add new language

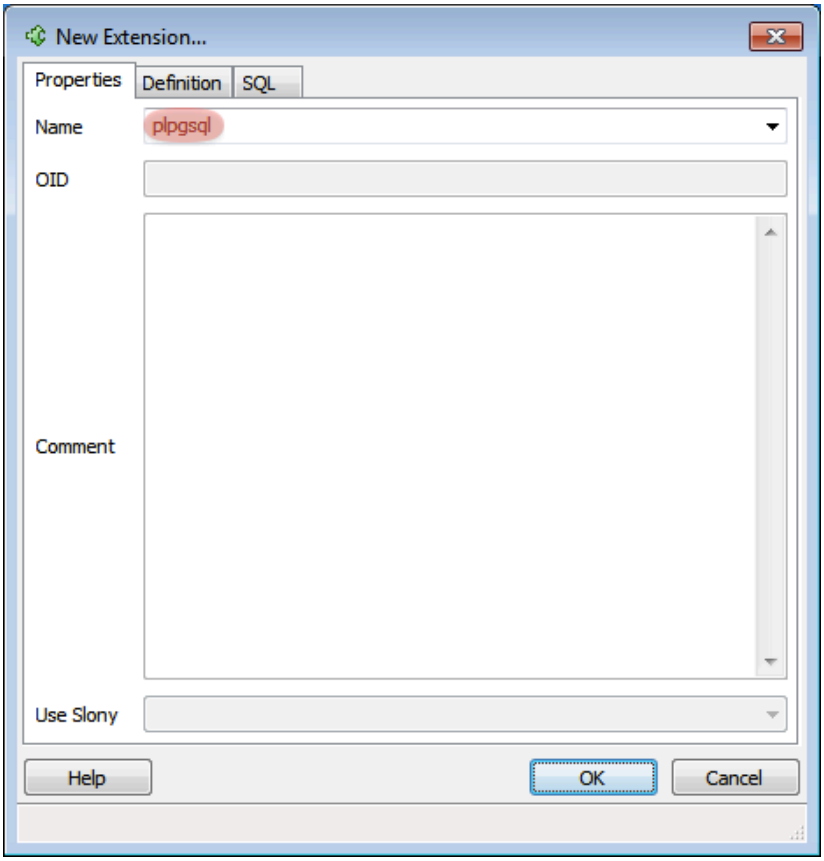


Figure 7.14: New language name

The OMERO.server also accesses the `omero\tmp` and `omero\log` folders of **the user account running the server process**. As the server makes heavier use of these folders than the clients, if that user's home folder is stored on a network the server can be slowed down. To get around this for the OMERO.server you can define an `OMERO_TEMPDIR` environment variable pointing to a temporary directory located on the local file system e.g. `C:\tmp\`.

7.1.6 Installation

- Download and extract the OMERO.server zip file, and note its location. Below it is referred to as `C:\OMERO.server`.
- Optionally, review `C:\OMERO.server\etc\omero.properties` which contains all default settings.

  You will need to open the file with a text editor. Do not edit the file. Any configuration settings you would like to change can be changed in the next step.
- Change any settings that are necessary using `bin\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 link[23]).

```
C:\> cd C:\OMERO.server
C:\OMERO.server\> bin\omero config set omero.db.name omero_database
C:\OMERO.server\> bin\omero config set omero.db.user db_user
C:\OMERO.server\> bin\omero config set omero.db.pass db_password
```

- If you have chosen a non-standard *OMERO binary repository* location above, be sure to configure the `omero.data.dir` property.

  When using `C:\` style file paths it is necessary to "escape" the backslashes. For example:

```
C:\> bin\omero config set omero.data.dir C:\\OMERO
```

- Create the OMERO database initialization script. You will be asked for the version of the script which you would like to generate, where both defaults can be accepted. Finally, you will be asked to enter and confirm password for your newly created OMERO root user.

  > **Warning:** For illustrative purposes, the default password for the OMERO rootuser is `root_password`. However, you should not use this default value for your installation but use your own choice of password instead.
  > This should **not** be the same password as your Linux/Mac/Windows root user!

```
C:\> cd C:\OMERO.server
C:\OMERO.server> bin\omero db script
Please enter omero.db.version [OMERO4.4]:
Please enter omero.db.patch [0]:
Please enter password for new OMERO root user: # root_password
Please re-enter password for new OMERO root user: # root_password
Saving to C:\OMERO.server\OMERO4.4__0.sql
```

  The generated SQL file is found in the folder where you called the "omero db script" command. This could cause a permission denied error in the populate db step if the postgres user cannot access that location. Move the file to a different location or use the -f option.
- Initialize your database with the script.
  1. Launch *SQL Shell (psql)* from the Start Menu under *Programs → PostgreSQL 9.1 → SQL Shell (psql)*

```
Server [localhost]:
Database [postgres]: omero_database
Port [5432]:
Username [postgres]: db_user
Password for user db_user:
Welcome to psql 9.1.4, the PostgreSQL interactive terminal.

Type:  copyright for distribution terms
       h for help with SQL commands
       ? for help with psql commands
       g or terminate with semicolon to execute query
       q to quit
```

[23] http://www.openmicroscopy.org/community/viewtopic.php?f=5&t=360#p922

```
Warning: Console code page (437) differs from Windows code page
         (1252) 8-bit characters might not work correctly. See psql
         reference page ''Notes for Windows users'' for details.
```

2. Execute the following to populate your database (**the forward slashes are intentional** - if you get a permission denied error it is because the path is wrong, not the slashes):

```
omero=> \i C:/OMERO.server/OMERO4.4__0.sql
...
...
omero=> \q
```

- Before starting the OMERO.server, you should run the OMERO diagnostics script so that you check that all of the settings are correct, e.g.

```
C:\OMERO.server\> bin\omero admin diagnostics
```

The diagnostic tool may say that psql is not found. This should not be a problem but you can fix it by adding its bin folder to the path. For example, `C:\Program Files (x86)\PostgreSQL\9.2\bin`. Remember to reboot after changing the environment.

- You can now start the server using:

```
C:\OMERO.server> bin\omero admin start
Creating C:\OMERO.server\var\master
Creating C:\OMERO.server\var\registry
No descriptor given. Using etc\grid\windefault.xml
Installing OMERO.master Windows service.
Successfully installed OMERO.master Windows service.
Starting OMERO.master Windows service.
Waiting on startup. Use CTRL-C to exit
...
```

If you have chosen a non-default install directory (other than `C:\OMERO.server`), the output will look like this:

```
C:\OMERO.server> bin\omero admin start
Found default value: C:\OMERO.server\var\master
Attempting to correct...
Converting from C:\OMERO.server to C:\OMERO.server
Changes made: 6
No descriptor given. Using etc\\grid\\windefault.xml
...
```

- If you would like to move the directory again, see `bin\winconfig.bat --help`, which gets called automatically on an initial install.
- You can now test that you can log-in as "root", either with the OMERO.insight client or command-line:

```
C:\OMERO.server> bin\omero login
Server: [localhost]
Username: [root]
Password:         # root_password
```

7.1.7 OMERO.web and administration

Note: In order to deploy OMERO.web in a production environment such as Apache or IIS please follow the instructions under *OMERO.web deployment*.

Otherwise, the internal Django webserver can be used for evaluation and development. In this case, you need to turn debugging on, in order that static files can be served by Django:

```
c:\OMERO.server> bin\omero config set omero.web.application_server development
c:\OMERO.server> bin\omero config set omero.web.debug True
c:\OMERO.server> bin\omero config set omero.web.session_engine "django.contrib.sessions.backends.cache"
c:\OMERO.server> bin\omero config set omero.web.cache_backend "file://C:/windows/temp/"
```

then start by

```
c:\OMERO.server> bin\omero web start
Starting django development webserver...
Validating models...
0 errors found

Django version 1.1.1, using settings 'omeroweb.settings'
Development server is running at http://0.0.0.0:4080/
Quit the server with CONTROL-C.
```

Once you have deployed and started the server you can use your browser to access the OMERO.web interface:

- http://localhost:4080/

Figure 7.15: OMERO.webadmin login

7.1.8 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

You should 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[24]. See the Feature list[25] for more advanced features you may want to use, and *Advanced configuration* on how to get the most out of your server.

JVM memory settings

The most likely change you will need to make to your application descriptors is increasing the memory settings. This is not done by default since it would prevent starting the server on some sites' test instance, but for production use a setting higher than 512MB is **highly** recommended. You can edit the templates.xml[26] file manually using a notepad-like editor. You should change each occurrence of `Xmx512M` to `Xmx2048M`; similarly `XX:MaxPermSize=128m` should be changed to `XX:MaxPermSize=256M`.

See the grid configuration section in the *Advanced server configuration* documentation for more information on *grid/templates.xml*.

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.

Troubleshooting

If you encounter a problem which is not addressed by the *Troubleshooting OMERO* page, you can post a message to our ome-users[27] mailing list as discussed on the *Community support* page.

Please include the output of the diagnostics command when asking for help with your server installation:

```
C:\OMERO.server> bin\omero admin diagnostics
================================================================================
OMERO Diagnostics 4.4.12
================================================================================

Commands:   java -version                  1.6.0     (C:\WINDOWS\system32\java.EXE --
3 others)
Commands:   python -V                      2.5       (C:\Python25\python.EXE)
Commands:  icegridnode --version       3.3     (C:\Ice-3.3.1\bin\x64\icegridnode.EXE -
- 2 others)
Commands:       icegridadmin  --version                    3.3               (C:\Ice-
3.3.1\bin\x64\icegridadmin.EXE -- 2 others)
Commands:  psql --version           8.3     (C:\Program Files (x86)\PostgreSQL\8.3\bin\psql.EX
- 2 others)

Server:     icegridnode                    running
Server:     Blitz-0                        active (pid = 7704, enabled)
Server:     DropBox                        active (pid = 8008, enabled)
Server:     FSServer                       active (pid = 7088, enabled)
Server:     Indexer-0                      active (pid = 4728, enabled)
Server:     OMERO.Glacier2                 active (pid = 5456, enabled)
Server:     OMERO.IceStorm                 active (pid = 800, enabled)
Server:     Processor-0                    active (pid = 7316, enabled)
Server:     Tables-0                       active (pid = 4420, enabled)
Server:     TestDropBox                    inactive (enabled)
Server:     Web                            inactive (enabled)
Server:     OMERO.master                   active (running as LocalSystem)
```

[24] http://cvs.openmicroscopy.org.uk/snapshots/movies/omero-4-1/mov/FlexPreview4.1-configuration.mov
[25] http://www.openmicroscopy.org/site/products/omero/feature-list
[26] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/templates.xml
[27] http://lists.openmicroscopy.org.uk/mailman/listinfo/ome-users

```
Log dir:    C:\hudson\trunk\dist\var\log   exists

Log files:  Blitz-0.log                   10.0 MB       errors=4    warnings=26
Log files:  DropBox.log                   2.0 KB
Log files:  FSServer.log                  1.0 KB
Log files:  Indexer-0.log                 8.0 MB        errors=18   warnings=1870
Log files:  OMEROweb.log                  n/a
Log files:  Processor-0.log               0.0 KB
Log files:  Tables-0.log                  0.0 KB
Log files:  TestDropBox.log               n/a
Log files:  master.err                    0.0 KB
Log files:  master.out                    0.0 KB
Log files:  Total size                    18.94 MB
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

7.2 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. Its structure is based on OMEIS[a].

[a] http://www.openmicroscopy.org/site/support/previous/ome-server/system-overview/ome-image-server/

7.2.1 Layout

The repository is internally laid out as follows:

```
C:\OMERO
C:\OMERO\Pixels      <--- Pixel data
C:\OMERO\Files       <--- Original file data
C:\OMERO\Thumbnails  <--- Thumbnail data
C:\OMERO\FullText    <--- Lucene full text search index
```

Your repository is not:

- the "database"
- the directory where your OMERO.server binaries are
- the directory where your OMERO.client (OMERO.insight, OMERO.editor or OMERO.importer) binaries are
- your PostgreSQL data directory

7.2.2 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 NullPointerExceptions, "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.

7.2.3 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. Remember that `C:\` style paths must have backslashes escaped. We strongly discourage the use of network mapped drives as locations for either the binary repository or the OMERO.server installation.

Your repository location can be changed from its `C:\OMERO` default by modifying your OMERO.server configuration as follows:

```
C:\> cd C:\OMERO.server
C:\OMERO.server\> bin\omero config set omero.data.dir D:\\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:

```
C:\> cd C:\OMERO.server
C:\OMERO.server\> bin\omero admin stop
C:\OMERO.server\> move C:\OMERO D:\
C:\OMERO.server\> bin\omero config set omero.data.dir D:\\OMERO
C:\OMERO.server\> bin\omero admin start
```

7.2.4 Access Permissions

Your repository should be owned or accessible by the same user that is starting your OMERO.server instance which may be different from the user you use to start OMERO. See *OMERO.server Windows Service* for more information.

To modify the access permissions to the binary repository, the `OMERO` folder properties can be accessed and the permissions settings changed (see *Repository Folder Permissions*). Another option (useful for batch permission changes) is the `icacls` (Windows 7) / `cacls` (Windows XP) command line utility. Please note that the new permissions will appear as *Special Permissions* in the *Security* tab when viewing folder properties. An example invocation allowing the user *omeservice* to read (*R*) and write (*W*) from and to the `OMERO` directory:

```
C:\>icacls OMERO /grant omeservice:RW
processed file: OMERO
Successfully processed 1 files; Failed processing 0 files
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

7.3 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.

For Windows-specific installation instructions, first see *OMERO.server installation*.

7.3.1 Ensuring you have a valid PostgreSQL version

For OMERO 4.4, PostgreSQL 8.4 or higher is required.

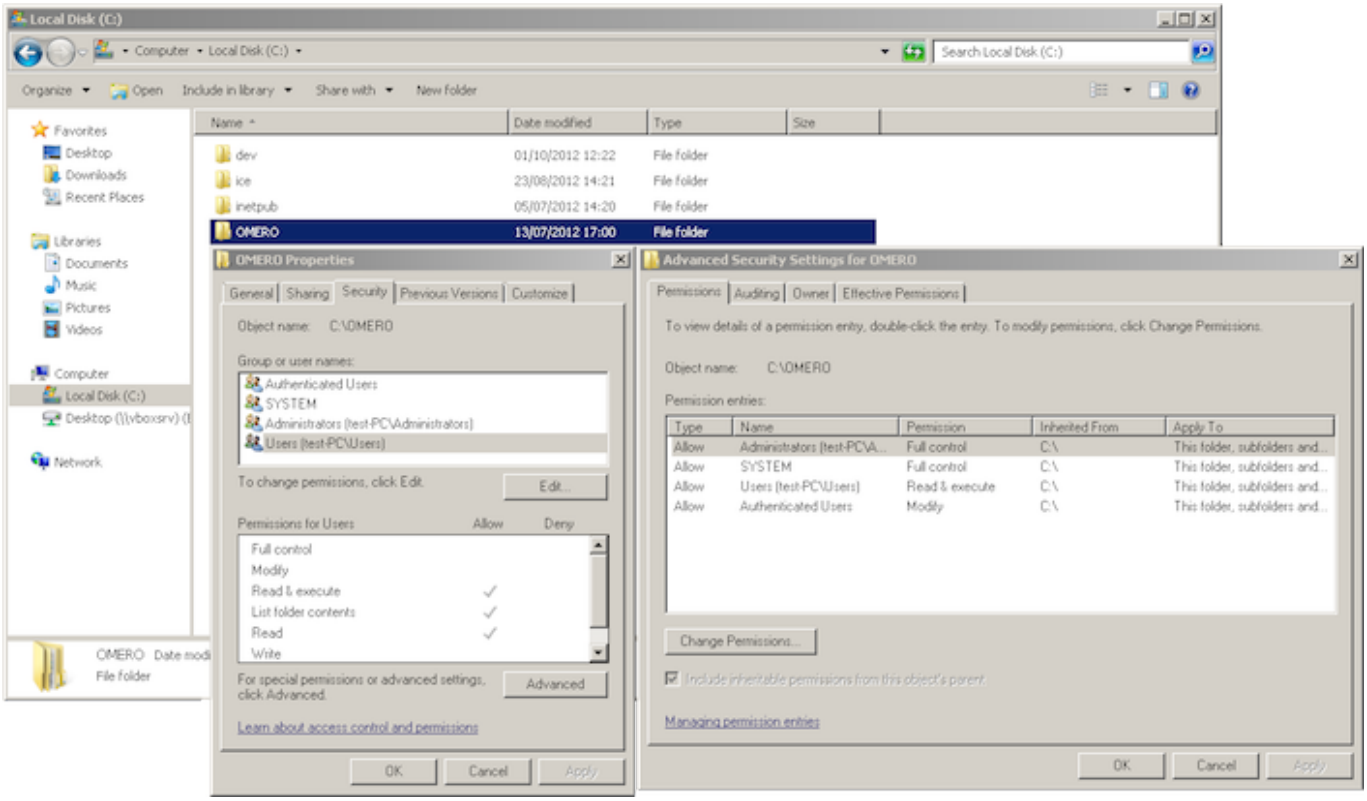


Figure 7.16: Repository Folder Permissions

If your default PostgreSQL installation is version 8.3 or earlier, you will need to upgrade to a more up-to-date version. We suggest the installer from EnterpriseDB[28]. Versions 8.4, 9.0 and 9.1 are known to work with OMERO 4.4; 9.1 is recommended.

Compatibility matrix

Versions of PostgreSQL which are compatible with OMERO are shown in the table below.

PostgreSQL	OMERO 4.1	OMERO 4.2	OMERO 4.3	OMERO 4.4
7.4	YES	NO *[1]*	NO *[1]*	NO *[4]*
8.1	YES	NO *[3]*	NO *[1]*	NO *[4]*
8.2	YES	YES	NO *[3]*	NO *[4]*
8.3	YES	YES	YES	NO *[4]*
8.4	YES	YES	YES	YES
9.x	YES *[2]*	YES *[2]*	YES *[2]*	YES

[1] Not suggested; see #4902[29]

[2] Configuration may be necessary; see #5662[30]

[3] Not suggested; see #5861[31]

[4] Unsupported; see #7813[32]

7.3.2 Checking PostgreSQL port listening status

You can check if PostgreSQL is listening on the default port (`TCP/5432`) by running the following command:

[28] http://www.enterprisedb.com/
[29] http://trac.openmicroscopy.org.uk/ome/ticket/4902
[30] http://trac.openmicroscopy.org.uk/ome/ticket/5662
[31] http://trac.openmicroscopy.org.uk/ome/ticket/5861
[32] http://trac.openmicroscopy.org.uk/ome/ticket/7813

```
C:\> netstat -an | find "5432"
```

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.

Once you have found the location of the `postgresql.conf` file on your particular installation, you will need to enable TCP listening. For PostgreSQL 8.4 and 9.x, the area of the configuration file you're 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
```

7.3.3 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
```

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**.

See also:

PostgreSQL[33] Interactive documentation for the current release of PostgreSQL.

Connections and Authentication[34] Section of the PostgreSQL documentation about configuring the server using *postgresql.conf*.

Client Authentication[35] Chapter of the PostgreSQL documentation about configuring client authentication with *pg_hba.conf*.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

7.4 OMERO.web deployment

OMERO.web is the web application component of the OMERO platform which allows for the management, visualization (in a fully multi-dimensional image viewer) and annotation of images from a web browser. It also includes OMERO.webadmin for managing users and groups.

OMERO.web is an integral part of the OMERO platform and can be deployed with:

- IIS 5.1, 6.0 or 7.0 on Microsoft Windows (since OMERO 4.2.1)
- FastCGI using a FastCGI capable web server such as Apache[36] (with mod_fastcgi[37] enabled), nginx[38] or lighttpd[39] (since OMERO 4.2.1)
- The built-in Django lightweight development server (for **testing** only)

You can find more information about FastCGI and where to get modules or packages for your distribution on the FastCGI website[40].

If you need help configuring your firewall rules, see the *Server security and firewalls* page.

7.4.1 Prerequisites

- OMERO and its prerequisites (see *OMERO.server installation*).
- Python version from 2.4 to 2.7 (due to backwards incompatibilities in Python 3.0, Django 1.3 does not work with Python 3.0; for more information see the Django Installation page[41]).
  - Python Imaging Library[42] should be available for your distribution
  - Matplotlib[43] should be available for your distribution.
- A FastCGI capable web server

7.4.2 Configuring OMERO from the command line

Configuration options can be set using the `omero config set` command:

```
C:\omero_dist>bin\omero config set <parameter> <value>
```

When supplying a value with spaces or multiple elements, use **double quotes**. The quotes will not be saved as part of the value (see below). Please use the **escape sequence** `\"` for nesting double quotes (e.g. `"[\"foo\", \"bar\"]"`).

To remove a configuration option (to return to default values where mentioned), simply omit the value:

```
C:\omero_dist>bin\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:

```
C:\omero_dist>bin\omero config get
```

which should return values without quotation marks.

A final useful option of `omero config edit` is:

```
C:\omero_dist>bin\omero config edit
```

which will allow for editing the configuration in a system-default text editor.

[36] http://httpd.apache.org/
[37] http://www.fastcgi.com/drupal/
[38] http://nginx.org/
[39] http://www.lighttpd.net/
[40] http://www.fastcgi.com/drupal/node/3
[41] https://docs.djangoproject.com/en/1.3/intro/install/
[42] http://www.pythonware.com/products/pil/
[43] http://matplotlib.org/

7.4.3 Quick start

Using IIS

Once you have IIS installed on your system, a straightforward set of steps is required to get the ISAPI WSGI[44] handler for OMERO.web working with your IIS deployment.

- Choose between FastCGI TCP (recommended) or FastCGI (advanced):

```
C:\omero_dist>bin\omero config set omero.web.application_server "fastcgi" / "fastcgi-tcp"
```

- Ensure that the ISAPI for IIS options are installed
- Download and install an ISAPI WSGI Installer[45] (we suggest the *Windows Installer*)
- For extended compatibility with multiple IIS versions ISAPI WSGI uses the IIS 5/6 WMI interface to interact with your IIS deployment. If you are using IIS 7 you must enable the IIS 6 WMI backwards compatibility options, as shown on the figure:

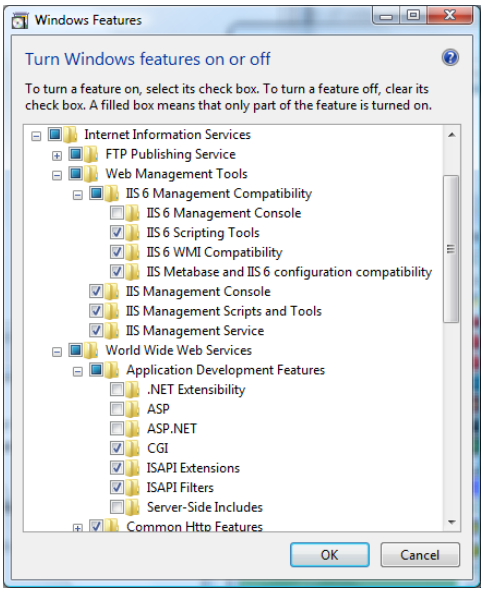


Figure 7.17: IIS 7 configuration options

- OMERO.web and ISAPI WSGI are **32-bit** applications on Windows at present. If you are attempting to run OMERO.web on a 64-bit version of Windows, you must enable 32-bit compatibility in the *Advanced Settings...* for the *Application Pool* assigned to your default *Site*. You can do this in the *IIS Manager* as follows:
- Configure OMERO.web bindings for IIS

```
C:\omero_dist>bin\omero config set omero.web.session_engine "django.contrib.sessions.backends.cache
C:\omero_dist>bin\omero config set omero.web.cache_backend "file://C:/windows/temp/"
C:\omero_dist>bin\omero web iis
```

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, turn Debugging on and start the server up:

[44] http://code.google.com/p/isapi-wsgi/
[45] http://code.google.com/p/isapi-wsgi/downloads/list

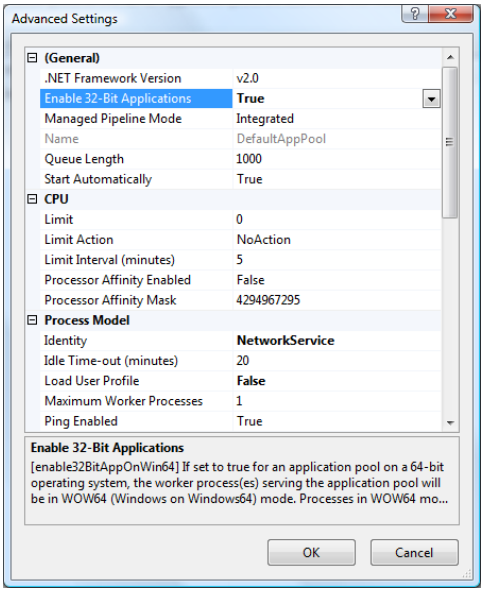


Figure 7.18: IIS 7 Application Pool Advanced Settings

```
C:\omero_dist>bin\omero config set omero.web.application_server development
C:\omero_dist>bin\omero config set omero.web.debug True
C:\omero_dist>bin\omero web start
Copying 'C:\omero_dist\lib\python\omeroweb\feedback\static\feedback\css\layout.css'
.....
Copying 'C:\omero_dist\lib\python\omeroweb\webstart\static\webstart\img\icon-omero-web.png'

735 static files copied to 'C:\omero_dist\lib\python\omeroweb\webstart\static'.
Starting OMERO.web... Validating models...

0 errors found
Django version 1.3.1, using settings 'omeroweb.settings'
Development server is running at http://0.0.0.0:4080/
Quit the server with CONTROL-C.
```

7.4.4 Logging in to OMERO.web

Once you have deployed and started the server, you can use your browser to access OMERO.webadmin or the OMERO.webclient:

- **http://your_host/omero** OR, for development server: **http://localhost:4080**

Figure 7.19: OMERO.webadmin login

Note: This starts the server in the foreground. It is your responsibility to place it in the background, if required, and manage its shutdown.

7.4.5 Customizing your OMERO.web installation

Note: Please use double quotes instead of single quotes and a proper escape sequence to specify options with multiple values.

Note: For clarity, some edge-case/in-development options may not be documented below. For the full list see:

```
C:\omero_dist>bin\omero web -h
```

OR look in lib/python/omeroweb/settings.py

- A list of servers the Web client can connect to. Default: `[["localhost", 4064, "omero"]]`.

  ```
  C:\omero_dist>bin\omero config set omero.web.server_list "[[\"prod.example.com\", 4064, \"prod\"],
  ```

- Email server and notification:
  - (**REQUIRED**) From : address to be used when sending e-mail. Default: `root@localhost`

    ```
    C:\omero_dist>bin\omero config set omero.web.server_email "webmaster@example.com"
    ```

  - (**REQUIRED**) Mail server hostname. Default: `localhost`.

    ```
    C:\omero_dist>bin\omero config set omero.web.email_host "email.example.com"
    ```

  - Mail server login username. Default: '' (Empty string).

    ```
    C:\omero_dist>bin\omero config set omero.web.email_host_user "username"
    ```

  - Mail server login password. Default: '' (Empty string).

    ```
    C:\omero_dist>bin\omero config set omero.web.email_host_password "password"
    ```

  - Mail server port. Default: `25`.

    ```
    C:\omero_dist>bin\omero config set omero.web.email_host_port "2255"
    ```

  - Use TLS when sending e-mail. Default: `False`.

    ```
    C:\omero_dist>bin\omero config set omero.web.email_use_tls "True"
    ```

  - Subject prefix for outgoing e-mail. Default: `"[Django] "`.

    ```
    C:\omero_dist>bin\omero config set omero.web.email_subject_prefix "Subject prefix for outgoing e-
    ```

- Controlling displayed scripts:

  - Since OMERO 4.3.2, OMERO.web has the ability to dynamically display scripts in the script runner menu just like OMERO.insight. Some scripts were not suitable for display initially and are excluded from the menu. You may wish to control which scripts your users can see in OMERO.web using this configuration option. Default: `'["\omero\figure_scripts\Movie_Figure.py", "\omero\figure_scripts\Split_View_Figure.py", "\omero\figure_scripts\Thumbnail_Figure.p "\omero\figure_scripts\ROI_Split_Figure.py", "\omero\export_scripts\Make_Movie.py"]'`

```
C:\omero_dist>bin\omero config set omero.web.scripts_to_ignore '[]'
C:\omero_dist>bin\omero config set omero.web.scripts_to_ignore '["\omero\my_scripts\really_bugg
```

- Enabling a public user:
  - Since OMERO 4.4.0, OMERO.web has the ability to automatically log in a public user.
    * First, create a public user. You can use any username and password you wish. If you do not want this user to be able to modify any of the data they see, you should put this user in a Read-Only group and the public data should be owned by another member(s) of this group. Now you can configure the public user:
    * Enable and disable the OMERO.web public user functionality. Default: `False`.

```
C:\omero_dist>bin\omero config set omero.web.public.enabled True
```

    * Set a URL filter for which the OMERO.web public user is allowed to navigate. Default: `^/(?!webadmin)` (Python regular expression[46]). You probably do not want the whole webclient UI to be publicly visible (although you could do this). The idea is that you can create the public pages yourself (see *OMERO.web framework* since we do not provide public pages. E.g. to allow only URLs that start with '/my_web_public' you would use:

```
C:\omero_dist>bin\omero config set omero.web.public.url_filter '^/my_web_public'

C:\omero_dist>bin\omero config set omero.web.public.url_filter'^/(my_web_public|webgateway)'
```

      Exotic matching techniques can be used but more explicit regular expressions are needed when attempting to filter based on a base URL:

```
'webtest' matches '/webtest' but also '/webclient/webtest'
'dataset' matches '/webtest/dataset' and also '/webclient/dataset'
'/webtest' matches '/webtest…' but also '/webclient/webtest'
'^/webtest' matches '/webtest…' but not '/webclient/webtest'
```

    * Server to authenticate against. Default: `1` (the first server in `omero.web.server_list`)

```
C:\omero_dist>bin\omero config set omero.web.public.server_id 2
```

    * Username to use during authentication. Default: `Not set.` (required if `omero.web.public.enabled=True`):

```
C:\omero_dist>bin\omero config set omero.web.public.user '__public__'
```

    * Password to use during authentication. Default: `Not set.` (required if `omero.web.public.enabled=True`):

```
C:\omero_dist>bin\omero config set omero.web.public.password 'secret'
```

[46] http://docs.python.org/2/library/re.html

- Administrator e-mail notification:
  - Admins list of people who get code error notifications. When debug mode is off and a view raises an exception, Django will e-mail these people with the full exception information. Default: `[]` (Empty list).

```
C:\omero_dist>bin\omero config set omero.web.admins '[["Dave", "dave@example.com"], ["Bob", "bo
```

- Ping interval:
  - Since OMERO 4.4.0, OMERO.web now pings the server to keep your session alive when you are logged in and have an active browser window. The duration between these pings can be configured. Default: `60000.` (every 60 seconds)

```
C:\omero_dist>bin\omero config set omero.web.ping_interval 12000
```

- Debug mode:
  - A boolean that turns on/off debug mode. Default: `False`.

```
C:\omero_dist>bin\omero config set omero.web.debug "True"
```

- Configuring additional web apps:
  - The OMERO.web framework allows you to add additional Django apps. For an example with installation instructions, see webmobile[47]
  - Download or clone from the git repository into the /omeroweb/ directory, then run

```
C:\omero_dist>bin\omero config set omero.web.apps '["<app name>"]'
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

7.5 OMERO.server Windows Service

OMERO.server installs a Windows Service to make the startup of the software automatic at Windows boot time.

The `omero admin start` command creates and starts a Windows service. In turn, `omero admin stop` stops and deletes the OMERO Windows service. Therefore, once `omero admin start` succeeds, it is possible to use all the regular Windows utilities, like `sc.exe` or the Services Manager, to stop OMERO.server without having the service removed completely.

If required, the OMERO.server service can be run as a different user (by modifying the *Log On* settings of the Windows service).

7.5.1 Service *Log On* user

Default Windows Local System user

When no specific Windows user has been defined using `omero config set`, the OMERO.server starts as the *Local System* user. This user has enough permissions to access data on the local file system. In most circumstances that should allow the OMERO.server service to access data inside the binary repository.

Custom user

A custom user can be configured to run the OMERO.server service. You can configure the OMERO Windows user by setting *omero.windows.user* and *omero.windows.pass*:

[47] https://github.com/will-moore/webmobile/

```
C:\OMERO.server\> bin\omero config set omero.windows.user USERNAME
C:\OMERO.server\> bin\omero config set omero.windows.pass PASSWORD
```

Warning: Setting *omero.windows.pass* exposes your user password in the OMERO configuration.

The user credentials can also be specified on the command line when running `omero admin start`. The `-u` parameter value is the user name, while the value of `-w` corresponds to the user's password:

```
C:\OMERO.server\> bin\omero admin start -u omeservice -w password
```

You can verify that a different user has been set as the *Log On* user for the OMERO.server service by accessing the Windows Services Manager (see *Windows Service Log On User Settings*).

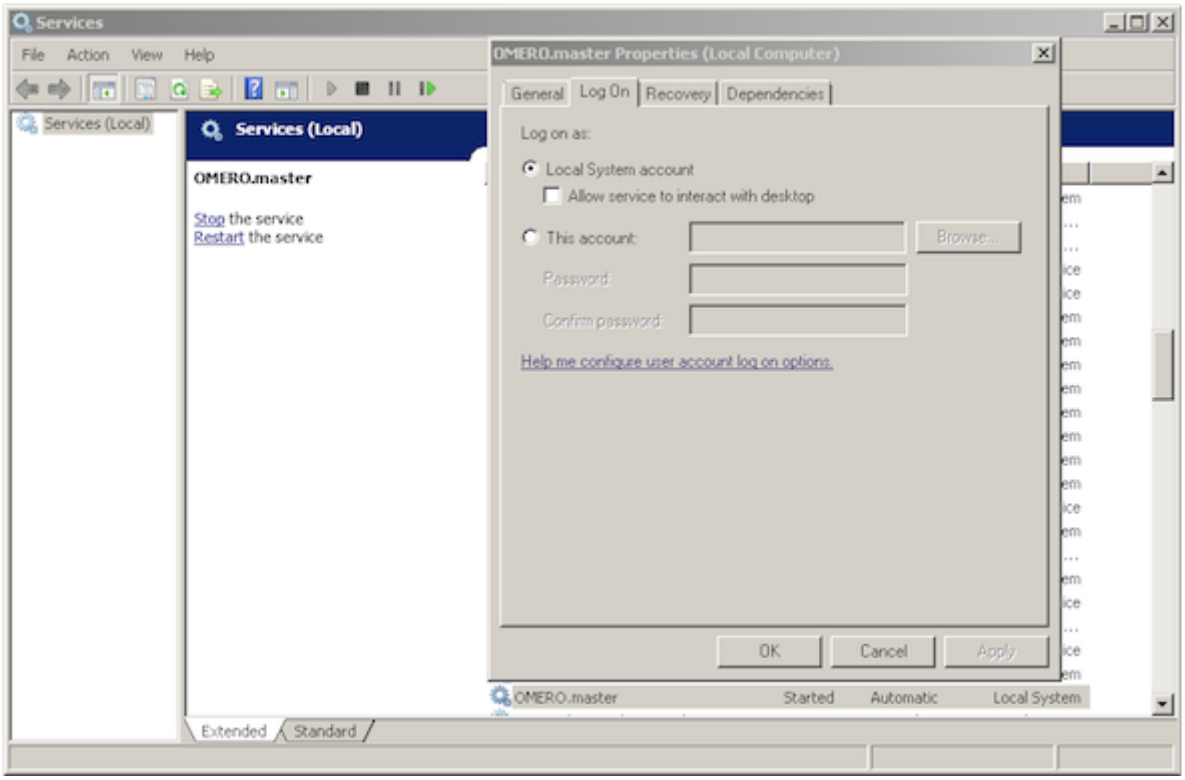


Figure 7.20: Windows Service *Log On* User Settings

7.5.2 Service startup mode

To start the Services Manager, simply navigate to *Start* → *All Programs* → *Accessories* → *Run* (Windows 7). In the dialog, type in `services.msc` and select *OK* (see *Run Windows Services Manager*).

This will bring up the Windows Services Manager. Here you can see the OMERO.master service running and also stop it. Additionally the *Log On* tab can be accessed here to configure under which user name the service is started (see *OMERO.server binary repository*).

It is also possible to change the service start-up type from *Automatic* to *Manual*. The automatic mode guarantees that OMERO.server will start during the Windows boot phase. Manual mode allows the logged in user or administrator to start the service after Windows has finished booting.

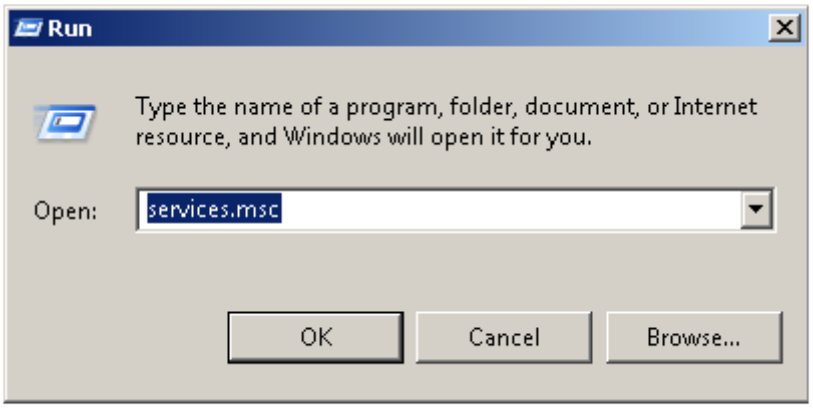


Figure 7.21: Run Windows Services Manager

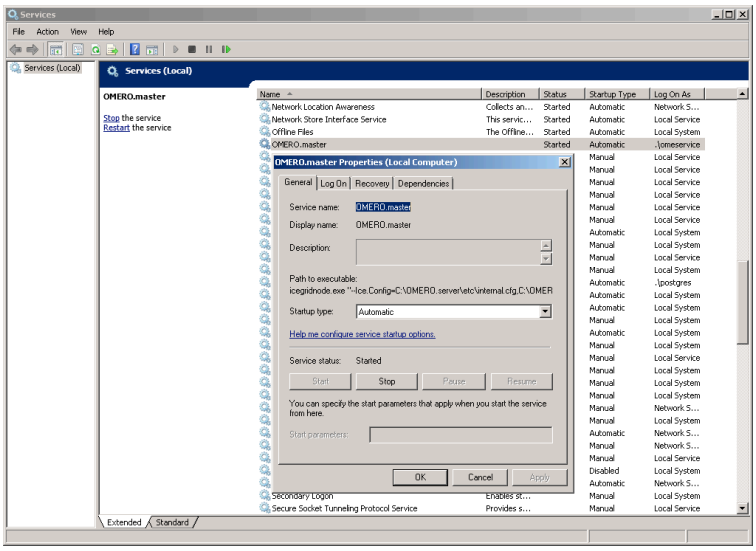


Figure 7.22: OMERO.master Service

7.5.3 Service events

Windows Event Viewer allows for watching events occurring in the OMERO.server service. To start the viewer, follow the same path as for Windows Services Manager, but this time type in `eventvwr.msc` (see *Starting Event viewer*).

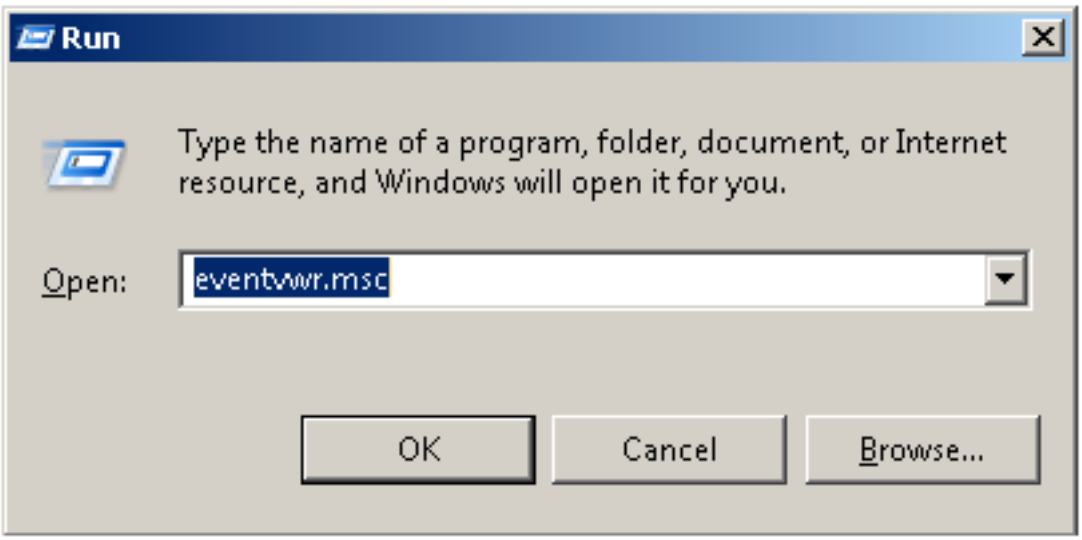


Figure 7.23: Starting Event viewer

The status events from OMERO.master will be registered in the *Application* view (though the log output from the server is in the configured directory).

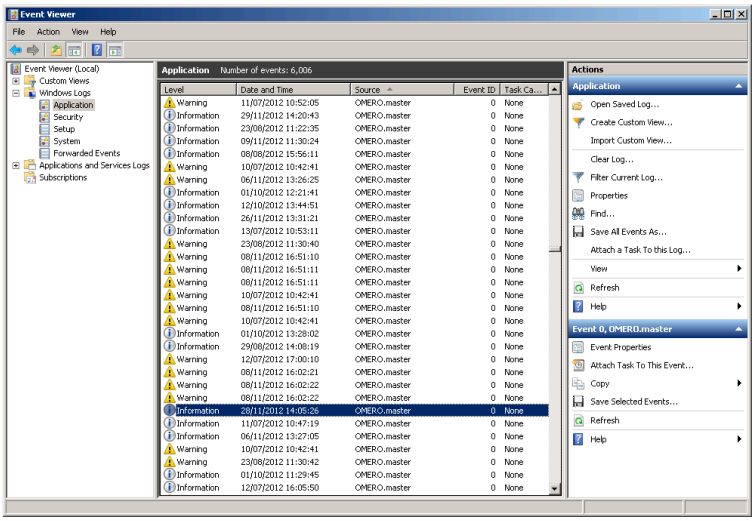


Figure 7.24: OMERO.master Events

CHAPTER

EIGHT

ADVANCED SERVER INSTALLATION

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

8.1 Troubleshooting OMERO

8.1.1 Frequently Asked Questions

If you cannot find what you are looking for here, try the FAQ[1].

8.1.2 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 "administrator-level user" on the *Windows installation page* and "root-level user" on the *UNIX installation page* (which includes Mac OS X). 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

[1] http://www.openmicroscopy.org/site/support/faq/

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 "omero.db.***" values during installation:

```
$ bin/omero config set omero.db.user db_user
$ bin/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:

```
$ bin/omero db script
Please enter omero.db.version [OMERO4.4]:
Please enter omero.db.patch [0]:
Please enter password for new OMERO root user:       # root_password
Please re-enter password for new OMERO root user:      # root_password
Saving to ~/OMERO4.4__0.sql
```

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**.

8.1.3 Server fails to start

1. Check that you are able to successfully connect to your PostgreSQL installation as outlined on the PostgreSQL page for your OS (*Windows PostgreSQL page* or *UNIX/Mac PostgreSQL page*).
2. Check the permissions on your `omero.data.dir` (`/OMERO` or `C:\OMERO` by default) as outlined on the *OMERO.server installation* page for Unix/Mac users, or the *OMERO.server binary repository* page for Windows users.
3. Are you on a laptop? If you see an error message mentioning "node master couldn't be reached"[2], 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"[3] or "could not lock file: var/registry/__Freeze/lock"[4], 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.

8.1.4 Remote clients cannot connect to OMERO installation

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. Please see the *Server security and firewalls* page.

[2] http://trac.openmicroscopy.org.uk/ome/ticket/7325
[3] http://trac.openmicroscopy.org.uk/ome/ticket/5576
[4] http://trac.openmicroscopy.org.uk/ome/ticket/7325

8.1.5 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 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
```

8.1.6 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 IceGrid configuration for your OMERO.server as follows:

- In `etc/grid/templates.xml`:

  …

  ```
  <server-template id="BlitzTemplate">
    <parameter name="index"/>
    <parameter name="config" default="default"/>
    <server id="Blitz-${index}" exe="java" activation="always" pwd="${OMERO_HOME}">
      <!--
      Debugging options:
      <option>-Xdebug</option>
      <option>-Xrunjdwp:server=y,transport=dt_socket,address=8787,suspend=n</option>
      -->
      <option>-Xmx512M</option>
      <option>-Djava.awt.headless=true</option>
      <option>-Dlog4j.configuration=${OMERO_ETC}log4j.xml</option>
      …
  ```

  Replace `-Xmx512M` initially with `-Xmx1024M` or greater as required, (use `-Xmx1024M -XX:MaxPermSize=128m` as an alternative).

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[5]. A 64-bit platform for your OMERO.server is **HIGHLY** recommended.

8.1.7 Import error when running bin/omero

```
Traceback (most recent call last):
File "bin/omero", line 67, in ?
```

[5] http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4724038

```
    import omero.cli
ImportError: No module named omero.cli
```

If you get any import related errors while running `bin/omero`, the most likely cause is that your PYTHONPATH is not properly set.

- If you installed Ice globally via your package manager, make sure you included ice-python.
- If you installed Ice manually, e.g. under `/opt/Ice-3.3.1` you need to add `/opt/Ice-3.3.1/python` (or similar) to your PYTHONPATH environment variable. See the Ice installation instructions for more information.

8.1.8 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"/>
```

8.1.9 OMERO.web issues

OMERO.web is not accessible from remote computer

To configure the out-of-the box setup to listen for webadmin and webclient connections on different host run:

```
c:\omero_dist> bin/omero web start 'host' 'port'
```

OMERO.web did not start on the production

The user opening OMEROweb.log files needs write permissions to the directory containing the log files. So, be sure you have a `log` directory with the correct ownership and the path set in `LOGDIR` matches the log directory.

- In your `/home/omero/omero_dist/var/lib/` directory add the following to your `custom_settings.py` file:
  - LOGDIR

    ```
    LOGDIR = '/home/omero/weblog/'
    ```

- Check if `/home/omero/omero_dist/var/lib/custom_settings.py` exists.
- Check who owns the log directory and log files:

```
$ ls -al /home/omero/weblog/
total 49
drwxr-xr-x  2 apache apache   120 Mar 31 11:29 .
drwxr-xr-x 10 apache apache   520 Mar 31 11:29 ..
-rw-r--r--  1 apache apache 23766 Mar 31 11:41 OMEROweb.log
-rw-r--r--  1 apache apache 23978 Mar 31 11:41 OMEROweb.log.2009-03-31
```

- or create log and database directories with, for example, `apache_user:apache_group` ownership:

```
mkdir /home/omero/weblog
chown apache_user:apache_group /home/omero/weblog
```

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[6].
- 'ImportError: No module named Image'. Install Python Imaging Library[7] (packages should be available for your distribution). Also double check if all of the prerequisites were installed from OMERO.web deployment (*UNIX instructions* or *Windows instructions*).

8.1.10 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
```

These warnings (also sometimes listed as ERRORS) can be safely ignored, and are solely related to how Java is installed on your system. See http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4751177 or this ome-users thread[8] on our mailing list for more information.

Too many open files

This 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 to occur. 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

[6] http://matplotlib.org/
[7] http://www.pythonware.com/products/pil/
[8] http://lists.openmicroscopy.org.uk/pipermail/ome-users/2009-March/001465.html

```
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
```

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[9] and Python Issue #3392[10] for more details.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

8.2 Server security and firewalls

8.2.1 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 (Secure Socket Layer)
- 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

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 any one of the mechanisms found on the community[11] page.

8.2.2 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[12] assigned ports for the Glacier2 router from ZeroC[13]. Both of these values, however, are completely up to you, see *SSL* below.

Important OMERO ports:

- **TCP/4063**
- **TCP/4064**

[9] http://trac.openmicroscopy.org.uk/ome/ticket/6201
[10] http://bugs.python.org/issue3392
[11] http://www.openmicroscopy.org/site/community/
[12] http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml
[13] http://www.zeroc.com

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
...
```

8.2.3 Passwords

The password hashes stored in the `password` table are generated equivalent to the command:

```
$ echo -n "ome" | openssl md5 -binary | openssl base64
vvFwuczAmpyoRC0Nsv8FCw==
```

If the password for the root user were lost, the only way to reset it (in the absence of other admin accounts) would be to manually update the password table.

```
$ PASS=`echo -n "ome" | openssl md5 -binary | openssl base64`

$ psql mydatabase -c " select * from password"
 experimenter_id |           hash
-----------------+--------------------------
               0 | Xr4ilOzQ4PCOq3aQ0qbuaQ==
(1 row)

$ psql mydatabase -c "update password set hash = '$PASS' where experimenter_id = 0"
UPDATE 1

$ psql mydatabase -c " select * from password"
 experimenter_id |           hash
-----------------+--------------------------
               0 | vvFwuczAmpyoRC0Nsv8FCw==
(1 row)
```

If you prefer, the `bin/omero` command can generate this update string for you:

```
Please enter password for new OMERO root user:
Please re-enter password for new OMERO root user:
UPDATE password SET hash = 'vvFwuczAmpyoRC0Nsv8FCw==' WHERE experimenter_id = 0;
$
```

8.2.4 Java key- and truststores.

If your server is connecting to another server over SSL, you will need to configure both a keystore and a truststore 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* (*Windows*).

- keystore path

```
bin/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.
```

- keystore password

```
bin/omero config set omero.security.keyStorePassword secret
```

- truststore path

```
bin/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.
```

- truststore password

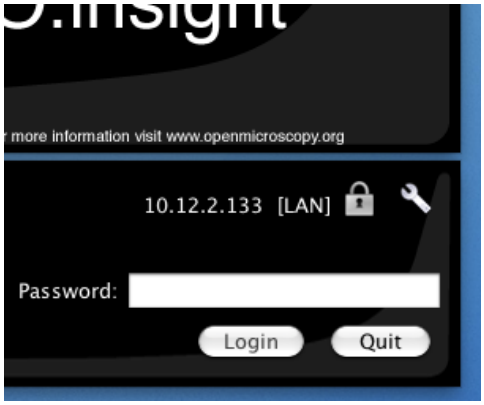
```
bin/omero config set omero.security.trustStorePassword secret
```

8.2.5 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, clients can request to have communication un-encrypted to speed up image loading 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 connection 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 and windefault.xml files, and as described above, default to 4064 and 4063 respectively, and can be modified with command:




```
$ bin/omero admin ports --help
usage: bin/omero admin ports [-h] [--prefix PREFIX] [--registry REGISTRY]
                             [--tcp TCP] [--ssl SSL] [--revert]

Allows modifying the ports from a standard OMERO install

To have two OMERO's running on the same machine, several ports must be modified from their default value
Internally, this command uses the omero.install.change_ports module.

Examples:

    bin/omero admin ports --prefix=1                              # sets ports to: 14061, 14063, 14064
    bin/omero admin ports --prefix=1 --revert                     # sets ports back to: 4061, 4063, 4064
    bin/omero admin ports --registry=4444 --tcp=5555 --ssl=6666  # sets ports to: 4444 5555 6666

Optional Arguments:
  In addition to any higher level options

  -h, --help           show this help message and exit
  --prefix PREFIX      Adds a prefix to each port ON TOP OF any other settings
  --registry REGISTRY  Registry port. (default: 4061)
  --tcp TCP            The tcp port to be used by Glacier2 (default: 4063)
  --ssl SSL            The ssl port to be used by Glacier2 (default: 4064
  --revert             Used to rollback from the given settings to the defaults
```

See also:

LDAP authentication

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

8.3 Advanced configuration

Overview

Describes configuration options that not everyone will need, but may be useful for optimizing, customizing, or monitoring your server.

The OMERO server provides several different configuration/extension points that you may want to make use of.

8.3.1 First step: configuration properties

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 and copying these properties is as easy as moving this file to a new server version.

The etc/omero.properties[14] 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, the `omero config` command is used to change those properties that you would like to customize.

Examples of doing this are on the main *Unix* and *Windows* pages, as well as the *LDAP installation* page.

Here we list some options which you are most likely to want to modify. See etc/omero.properties[15] for more details.

omero.sessions.timeout

omero.sessions.timeout sets the duration of inactivity in milli-seconds after which a login is required (default is 600000, or 10 minutes). To change the default, for example, to 1 hour use:

```
$ bin/omero config set omero.sessions.timeout 3600000
```

omero.db.poolsize

omero.db.poolsize sets the number of connections to PostgreSQL which will be used by OMERO (default is 10):

```
$ bin/omero config set omero.db.poolsize 50
```

Your database installation will need to be configured to accept *at least* as many, preferably more, connections as the value of "omero.db.poolsize"

8.3.2 Last resort: 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[16]. "default.xml" is used on Unix systems, and "windefault.xml" is used on Windows systems. Both, make use of "templates.xml".

Modifying the application descriptors

When you run `omero admin start` without any other arguments, it looks up the default **application descriptor** for your platform:

```
~/git/dist $ bin/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:

```
$ bin/omero admin start --help
usage: bin/omero admin start [-h] [-u USER] [file] [targets [targets ...]]

Start icegridnode daemon and waits for required components to come up, i.e. status == 0

             If the first argument can be found as a file, it will
```

[14] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/omero.properties
[15] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/omero.properties
[16] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/etc/grid

```
              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:
  file                  Application descriptor. If not provided, a default will be used
  targets                Targets within the application descriptor which should be activated.
```

If a file is passed in as the first argument, then that **application descriptor** as opposed to 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 activates the monitoring of the Blitz server via JMX. If you need to modify the "jmxport" or "jmxhost" variables, you will need to do so directly in the application descriptor XML.

8.3.3 Changing ports / multiple servers on a single host

Since changing all the references to port numbers (4061, 4063, 4064, etc) in the grid configuration can be cumbersome, a `omero admin` command is provided to make the modifications for you. See the *SSL* section of the *Server security and firewalls* page for more information.

By modifying the default OMERO ports, it is possible to run multiple OMERO servers on the same physical machine.

```
# First server
cd /usr/local/omero-4.2
bin/omero admin ports --prefix=1
bin/omero admin start
# Second server
cd /usr/local/omero-4.3
bin/omero admin ports --prefix=2
bin/omero admin start
```

Clients will need to use the appropriate port (either 14064 or 24064) to connect to OMERO.

8.3.4 Extending Omero

Finally, if configuration does not suffice, there are also options to extending OMERO with your own code. These are described on the development site under *Extending OMERO*.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

8.4 LDAP authentication

LDAP[17] 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.

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})
```

8.4.1 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.

[17] http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

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*.

8.4.2 LDAP properties

The LDAP plugin is configured via several configuration properties, all starting with `omero.ldap.`. The default values for these properties are set in the file `etc/omero.properties`. An overview of all the properties is outlined in *LDAP configuration overview*.

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
```

`omero.ldap.user_filter` will be AND'ed 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.

- 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.
- 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 NewUserGroupBean interface. See the developer documentation *LDAP plugin design* for more info.

Compound Filters

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.group_filter=(&(objectClass=groupOfNames)(mail=omero.flag))
```

This filter is valid and will cause the filter to 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
```

8.4.3 LDAP configuration overview

Like many pieces of OMERO.server configuration, LDAP-specific configuration is done by specifying extra properties during installation. The default values for the LDAP properties are listed in the `etc/omero.properties` file inside your OMERO installation directory.

Note: Please remember that once a change has been made, a server restart will be needed.

Change any settings that are necessary via bin/omero config.

- Enable or disable LDAP (true/false)

  ```
  bin/omero config set omero.ldap.config true
  ```

- LDAP server URL string

  ```
  bin/omero config set omero.ldap.urls ldap://ldap.example.com:389
  ```

  Note: A SSL URL above should look like this: ldaps://ldap.example.com:636

- LDAP server bind DN (if required; can be empty)

```
bin/omero config set omero.ldap.username cn=Manager,dc=example,dc=com
```

- LDAP server bind password (if required; can be empty)

```
bin/omero config set omero.ldap.password secret
```

- LDAP server base search DN

```
bin/omero config set omero.ldap.base dc=example,dc=com
```

- The filter applied to all users; can be empty in which case any LDAP user is valid

```
bin/omero config set omero.ldap.user_filter '(objectClass=inetOrgPerson)'
```

- LDAP referral options (defaults to "ignore"; available options are "ignore, "follow" or "throw" as per the JNDI referrals documentation[18])

```
bin/omero config set omero.ldap.referral follow
```

8.4.4 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 will need to configure a key and trust store for Java. See the *Server security and firewalls* page for more information.

8.4.5 Synchronizing LDAP on user login

This feature allows for LDAP to be considered the authority on user/group membership. With the following settings enabled each time a user logs in to OMERO their LDAP groups will be read from the LDAP server and reflected in OMERO. Enabling this will result in any bespoke OMERO groups that have been created being removed from the user's profile. The groups will still exist on the server but the association between user and group will not be reflected unless such a link is made in LDAP.

```
bin/omero config set omero.ldap.sync_on_login true
```

8.4.6 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:

```
bin/omero config set omero.security.password_provider chainedPasswordProvider
```

When the LDAP password provider implementation changes, previous versions can be configured as necessary.

[18] http://docs.oracle.com/javase/jndi/tutorial/ldap/referral/jndi.html

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:

```
bin/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:

```
bin/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?[19] Forum discussion if you have LDAP requirements that are not covered by the above configuration

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

8.5 Installing OMERO.tables

OMERO.tables provide a way to efficiently store large, tabular results within OMERO. If you would like to find out more about the use of the OMERO.tables API, see *OMERO.analysis*

8.5.1 Requirements

If you would like to help test the Tables API, you will need the following installed:

- HDF5[20]
- NumPy[21] points to downloads at http://sourceforge.net/projects/numpy/
- PyTables[22] (Some packages include HDF5)

8.5.2 Unix

PyTables is likely available from the package repository of your Unix-flavor. This includes Mac OS X (homebrew), Debian and Ubuntu (apt-get), Centos (yum), and SuSE (yast). Here we've shown manual instructions using virtualenv.

Manually

[20] http://www.hdfgroup.org/HDF5/release/obtain5.html
[21] http://numpy.sourceforge.net/numdoc/HTML/numdoc.htm
[22] http://pytables.github.com/downloads.html

```
$ virtualenv $HOME/virtualenv
$ uname -o -p
i686 GNU/Linux
$ gcc --version
gcc (GCC) 4.1.2 20080704 (Red Hat 4.1.2-44)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

$ wget http://www.hdfgroup.org/ftp/HDF5/current/src/hdf5-1.8.3.tar.gz
$ tar xzf hdf5-1.8.3.tar.gz
$ cd hdf5-1.8.3
$ ./configure --prefix=$HOME/virtualenv
$ make
$ make install
$ export LD_LIBRARY_PATH=$HOME/virtualenv/lib
$ . $HOME/virtualenv/bin/activate
$ easy_install tables
```

Checking that it works

After that, the following should succeed:

```
josh@mac:~$ python
Python 2.5.4 (r254:67916, Jun 24 2009, 20:23:29)
[GCC 4.0.1 (Apple Computer, Inc. build 5370)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tables
>>> tables.test()
--------------------------------------------------------------------------------
PyTables version:  2.1
HDF5 version:      1.8.3
NumPy version:     1.3.0
Zlib version:      1.2.3
BZIP2 version:     1.0.5 (10-Dec-2007)
Python version:    2.5.4 (r254:67916, Jun 24 2009, 20:23:29)
[GCC 4.0.1 (Apple Computer, Inc. build 5370)]
Platform:          darwin-i386
Byte-ordering:     little
…
```

Once the required Python libraries are installed, starting OMERO will automatically start up the OMERO.tables service; there should be no need for further configuration or interaction.

8.5.3 Windows

The following specific packages have been tested on Windows 7 Enterprise:

- PIL: http://effbot.org/media/downloads/PIL-1.1.7.win32-py2.6.exe
- SciPy: http://sourceforge.net/projects/scipy/files/scipy/0.11.0/scipy-0.11.0-win32-superpack-python2.6.exe/download
- NumPy: http://sourceforge.net/projects/numpy/files/NumPy/1.6.2/numpy-1.6.2-win32-superpack-python2.6.exe/download
- PyTables: http://www.lfd.uci.edu/~gohlke/pythonlibs/#pytables
- HDF (with szip and zlib): http://www.hdfgroup.org/ftp/HDF5/current/bin/windows/

After installing all the Windows prerequisites OMERO.tables should start up during the OMERO.server startup. It can be verified by looking at the output of `omero admin diagnostics`:

```
Server:     Tables-0                        active (pid = 3176, enabled)
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

8.6 OMERO.movie

A short decription on how to create movies from OMERO.

8.6.1 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.

8.6.2 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[23]. It should be noted that the DivX avi is normally 1/3 to 1/10 the size of the Quicktime movie.

8.6.3 Installing the make movie script

The make movie script currently uses the mencoder[24] utility to encode the movies, this command should be in the path of the computer (icegrid node) running the script.

You can find windows installs for mencoder at http://sourceforge.net/projects/mplayer-win32/files/

We have Mac OSX installs for mencoder[25] which were originally provided here[26]. 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 debs for mencoder.

Make movie also usesPython Imaging Library[27] and numpy[28].

8.6.4 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)

[23] http://www.divx.com/
[24] http://www.mplayerhq.hu/design7/dload.html
[25] http://cvs.openmicroscopy.org.uk/snapshots/mencoder/mac/
[26] http://stefpause.com/apple/mac/mplayer-os-x-10rc1-and-mencoder-binaries/
[27] http://www.pythonware.com/products/pil/
[28] http://www.scipy.org/Download

- splitView: Should we show the split view in the movie (not available yet)
- showTime: Show the average time of the aquisition 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)

8.6.5 Platform-specific notes

Windows

For Windows, you can download a "MPlayer-rtm-svm-<version>" bundle from http://sourceforge.net/projects/mplayer-win32/[29]. You will need 7zip[30] to open the bundle. Then you will need to add the directory containing "mencoder.exe" to your system path and restart.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

8.7 Installing new 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 new script sharing service[31].

8.7.1 Prerequisites

8.7.2 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[32], you can use the GitHub app for your OS[33] (available for Mac and Windows but not Linux). The basic workflow is:

- fork our omero-user-script[34] 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.

[29] http://sourceforge.net/projects/mplayer-win32/
[30] http://www.7-zip.org/download.html
[31] http://www.openmicroscopy.org/site/community/scripts
[32] http://www.openmicroscopy.org/site/support/contributing/using-git.html
[33] http://help.github.com/articles/set-up-git
[34] https://github.com/ome/omero-user-scripts

CHAPTER

NINE

SERVER MAINTENANCE

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

9.1 OMERO.server backup and restore

9.1.1 Cleaning up your binary repository

The OMERO.server does not remove files from disk until a cleanup task has been run. A script to do this is included in the OMERO.server distribution `lib/python/omero/util/cleanse.py` which can be used so:

```
$ bin/omero admin cleanse /OMERO
```

This can be performed daily using cron with a script such as:

```
#!sh
#!/bin/bash

USERNAME="root"
PASSWORD="root_password"
BINARY_REPOSITORY="/OMERO"
OMERO_PREFIX=/home/omero/OMERO-CURRENT
$OMERO_PREFIX/bin/omero -s localhost -u $USERNAME -w $PASSWORD admin cleanse $BINARY_REPOSITORY
```

9.1.2 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.

9.1.3 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/"/>
...
```

9.1.4 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 (*UNIX/Mac information page* or *Windows information page*; assumed to be `/OMERO` or `C:\OMERO`)
3. OMERO.server configuration

> **Note:** 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 now available under https://github.com/ome/scripts which provides help for merging your lib/scripts directories.

You should back up *(1)* and *(2)* regularly.

> **Warning: In the event of a catastrophic failure, no recovery of your OMERO.server metadata (users, trees, logins etc.) is possible unless you have a backup of your PostgreSQL database.**

You need to back up *(3)* only before you make changes. You can copy it into `/OMERO/backup` to ensure it is kept safe:

```
$ bin/omero config get > /OMERO/backup/omero.config
```

> **Note:** 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`.

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[1] *(Chapter 24. Backup and Restore)*.

> **Note:** Regular backups of your PostgreSQL database are vital as, in the event of a catastrophic failure, **no recovery of your complete OMERO.server setup is possible without one.**

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 regularly as this will, especially if this document's

[1] http://www.postgresql.org/docs/9.1/interactive/backup.html

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[2].

9.1.5 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: *downloads* <> page that **matches** the version you originally had installed, all that is required is to restore your backup preferences by running:

```
$ bin/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:

[2] http://www.nongnu.org/rdiff-backup/docs.html

```
$ 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[3] Wikipedia page listing the backup softwares.

PostgreSQL 9.1 Interactive Manual[4] Chapter 24: Backup and Restore

rdiff-backup documentation[5] Online documentation of rdiff-backup project

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

9.2 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 roadmap[6] for exactly what will go into a release. We always inform our mailing lists[7] of the development status. Finally, all the 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.

Note: Before starting the upgrade, please ensure that you have obtained all the prerequisites for installation, documented for Unix and Windows. In particular, ensure that you are running a suitable version of PostgreSQL to enable successful upgrading of the database.

If you encounter errors during a OMERO upgrade, database upgrade, etc. you should retain as much log information as possible and notify the OMERO.server team via the mailing lists available on the community[8] page.

See the full details of OMERO 4.4.12 features in the Announcements[9] forum.

For all users, the basic workflow for upgrading your OMERO.server is listed below. Please refer to each section for additional details.

- Perform a database backup
- Copy new binaries
- Upgrade your database
- Merge script changes
- Update your configuration
- Restart your database
- Restore a database backup

Warning: With 4.4.12, the default JDBC password provider has been modified to add password salting support. This implies that once a server has been upgraded and deployed, if passwords are modified, you will not be able to easily revert to a configuration without salting. To keep using the legacy password provider without salting support, you will need to configure `omero.security.password_provider` to use the legacy `chainedPasswordProviderNoSalt` as described in the *Legacy password providers* section.

[6] https://trac.openmicroscopy.org.uk/ome/roadmap
[7] http://www.openmicroscopy.org/site/community/
[8] http://www.openmicroscopy.org/site/community/
[9] http://www.openmicroscopy.org/community/viewforum.php?f=11

Warning: 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, make sure to replace the values of **db_user** and **omero_database** with the actual database user and database name for your installation.

9.2.1 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
```

9.2.2 Copy new binaries

Before copying the new binaries, stop the existing server:

```
$ cd OMERO.server
$ bin/omero web stop
$ bin/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-4.4.12-ice3x-byy.zip
$ ln -s OMERO.server-4.4.12-ice3x-byy OMERO.server
$ cp OMERO.server-old/etc/grid/config.xml OMERO.server/etc/grid
```

Note: `ice3x` and `byy` **need to be replaced** by the appropriate Ice version and build number of OMERO.server.

9.2.3 Upgrade your database

Warning: This section only concerns users upgrading from a 4.3 or earlier server. If upgrading from a 4.4 server, you do not need to upgrade the database.

Run the upgrade script

You **must** use the same username and password you have defined during *OMERO.server installation*. The 4.4 upgrade script should execute in a short time.

```
$ cd OMERO.server
$ psql -h localhost -U db_user omero_database < sql/psql/OMERO4.4__0/OMERO 4.3__0.sql
Password for user db_user:
...
...
                            status
-----------------------------------------------------------------
                                                                +
                                                                +
                                                                +
YOU HAVE SUCCESSFULLY UPGRADED YOUR DATABASE TO VERSION OMERO 4.4__0+
                                                                +
                                                                +

(1 row)
```

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 'REINDEX DATABASE ''omero_database'' FORCE;'
$ psql -h localhost -U db_user omero_database -c 'VACUUM FULL VERBOSE ANALYZE;'
```

9.2.4 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. In order to facilitate saving your work, we have turned the scripts into a Git submodule which can be found at https://github.com/ome/scripts.

For further information on managing your scripts, refer to *Installing new scripts*. If you require help, please contact the OME developers.

9.2.5 Update your configuration

Environment variables

If you changed the directory name where the 4.4.12 server code resides, make sure to update any system environment variables. Before restarting the server, make sure your PATH and PYTHONPATH system environment variables are pointing to the new locations.

JVM memory settings

If you modified your memory settings, these changes will be lost and you will need to update the memory settings for the new server. Refer to the *JVM memory settings* sub-section of the OMERO.server installation section (*Unix* or *Windows*) for more information.

Changes to OMERO.web URLs

In order to ease deployment and avoid errors for IIS (Windows production deployment) and Apache (notably CentOS/RHEL 5 and 6) OMERO.web now defaults to being "mounted on `/omero`". The new OMERO.web web server stanzas have redirects in them with the notable exception of IIS. Depending on your web server configuration you may need to visit your OMERO.web instance at

```
http://example.com/omero/
```

As a result of this your web server configuration stanza generated by the previous version of `bin/omero web config` has to be replaced with the new version. To generate the relevant configuration, please run `bin/omero web config <webserver>`, update and restart your web server.

9.2.6 Restart your database

- Following a successful database upgrade, you can start the server.

```
$ cd OMERO.server
$ bin/omero admin start
```

- If anything goes wrong, please send the output of `bin/omero admin diagnostics` to ome-users@lists.openmicroscopy.org.uk[10].
- Start OMERO.web with the following command:

[10] ome-users@lists.openmicroscopy.org.uk

```
$ bin/omero web start
```

9.2.7 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 -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.

```
$ bin/omero config set omero.db.name omero_from_backup
```

See also:

Legacy[11] Legacy part of the OME website containing upgrade instructions for previous versions of the OMERO server.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

9.3 OMERO upgrade checks

On each startup the OMERO server checks for available upgrades via the UpgradeCheck class[12]. An HTTP GET call is made to the URL configured in `etc/omero.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[13] for assistance.

9.3.1 Actions

Currently the only action taken when an upgrade is necessary is a log statement at WARN level.

```
2011-09-01 12:21:32,070 WARN  [               ome.system.UpgradeCheck] (     main) UP-
GRADE AVAILABLE:Please upgrade to 4.4.12 See http://trac.openmicroscopy.org.uk/omero for the lat
est version
```

Future versions may also send emails and/or IMs to administrators. In the case of critical upgrades, the server may refuse to start.

9.3.2 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

[12] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/system/UpgradeCheck.java
[13] http://www.openmicroscopy.org/site/support/bio-formats4/users/comlinetools/index.html#version-checker

- 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.

9.3.3 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=
```

9.3.4 Developers

To use the UpgradeCheck class from your own code, it is necessary to have `common.jar` on your classpath. Then,

```
ResourceBundle bundle = ResourceBundle.getBundle("omero")
String version = bundle.getString("omero.version");
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.

9.3.5 Updating the registry version after a release

```
$ psql -h localhost -U postgres feedback

feedback=# select * from registry_version;
 id |  version
----+-----------
  1 | Beta-4.2.2
(1 row)

feedback=# select now();
              now
-------------------------------
 2011-06-27 16:01:30.749654+01
(1 row)

feedback=# update registry_version set version = 'Beta-4.3.0' where id = 1;
UPDATE 1
```

See also:

OMERO.server installation Instructions for installing OMERO.server on UNIX & UNIX-like platforms

OMERO.server installation Instructions for installing OMERO.server on Windows platforms

OMERO.server upgrade Instructions for upgrading OMERO.server

Server security and firewalls Description of OMERO security practices

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

9.4 OMERO Command Line Interface

See also:

OMERO Command Line Interface User documentation for the Command Line Interface

OMERO Command Line Interface Developer Documentation for the Command Line Interface

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.

9.4.1 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:

```
$ bin/omero db script OMERO4 0 secretpassword
Using OMERO4 for version
Using 0 for patch
Using password from commandline
Saving to /omero/OMERO4__0.sql
$ psql omero < OMERO4__0.sql
```

9.4.2 Server configuration

The `omero config` command is responsible for reading / writing user-specific profiles stored under `etc/grid/config.xml`. To get the current profile, use the `omero config def` command:

```
$ bin/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`:

```
$ bin/omero config get

$ bin/omero config set example "my first value"

$ bin/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 bin/omero config def
another

$ OMERO_CONFIG=another bin/omero config get

$ OMERO_CONFIG=another bin/omero config set example "my second value"

$ OMERO_CONFIG=another bin/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 `etc/omero.properties`. To add several values to a configuration, you can pipe them via standard in using `omero config load`:

```
$ grep omero.ldap etc/omero.properties | OMERO_CONFIG=ldap bin/omero config load

$ OMERO_CONFIG=ldap bin/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 bin/omero config set omero.ldap.trustStore

$ OMERO_CONFIG=ldap bin/omero config set omero.ldap.trustStorePassword

$ OMERO_CONFIG=ldap bin/omero config set omero.ldap.keyStore

$ OMERO_CONFIG=ldap bin/omero config set omero.ldap.keyStorePassword

$ OMERO_CONFIG=ldap bin/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:

```
$ bin/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`:

```
$ bin/omero config drop
```

9.4.3 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:

```
$ bin/omero admin start
```

Server diagnostics

```
$ bin/omero admin diagnostics
```

9.4.4 User/group management

The `omero user` and `omero group` commands provide functionalities to add and manage users and groups on your database.

User creation

New users can be added to the database using the `omero user add` command:

```
$ bin/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 as a member of group 2 identified as jsmith, enter:

```
$ bin/omero user add jsmith John Smith 2
```

Additional parameters such as the email address, institution, middle name etc can be passed as optional arguments to the `omero user add` command.

Group creation

New groups can be added to the database using the `omero group add` command:

```
$ bin/omero group add -h
```

During the addition of the new group, you need to specify the name of the new group:

```
$ bin/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:

```
$ bin/omero group add read-only-1 --perms='rwr---'
$ bin/omero group add read-annotate-1 --type=read-annotate
```

See also:

Permissions overview Description of the three group permissions levels (private, read-only, read-annotate).

Lists of users/groups on the OMERO server can be queried using the `omero user list` and `omero group list` commands:

```
$ bin/omero user list
$ bin/omero group list
```

Group management

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
$ bin/omero group adduser jsmith --name=read-annotate-1
# Remove jsmith from group read-annotate-1
$ bin/omero group removeuser jsmith --name=read-annotate-1
# Add jsmith to group read-only-1
$ bin/omero user joingroup read-only-1 --name=jsmith
# Remove jsmith from group read-only-1
$ bin/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
$ bin/omero group adduser jsmith --name=read-annotate-1 --as-owner
# Remove jsmith from the owner list of group read-annotate-1
$ bin/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:

```
$ bin/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:

```
$ bin/omero group copyusers read-only-1 read-only-2
```

To copy the group owners, use the same command with the *`--as-owner`* optional argument:

```
$ bin/omero group copyusers read-only-1 read-only-2 --as-owner
```

CHAPTER

TEN

OTHER ADVANCED TOPICS

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

10.1 Permissions overview

In the 4.4 release of OMERO, the groups and permissions system has been revamped to allow users to share data with more control. **Users can now move data between groups that they are a member of**.

See also:

OMERO permissions history, querying and usage

10.1.1 Summary

A user may belong to one or more groups, and the data in a group may now **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 now in 4.4 the user can easily copy their data into another group.

10.1.2 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 than a standard group member, including the ability to add other members to the group.

Group member This is the standard user.

Groups and users must be created by the server administrator. Users can then be added by the administrator or by one of the group owners assigned by the administrator. This would typically be the PI of the lab. The group's owners or server administrator can also choose the permission level for that group. See the OMERO.insight and OMERO.web admin movies below for more information about groups and how to administrate them in OMERO.

See also:

OMERO.insight Admin update in OMERO 4.4[1] Movie describing the administration tools update under OMERO.insight for OMERO 4.4

OMERO.web Admin update in OMERO 4.4[2] Movie describing the administration tools update under OMERO.web for OMERO 4.4

10.1.3 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:

- This group would be designed so that a PI as *Group owner* and their student, as a *Group member*, can access the student's data. A student might use this as somewhere to store all of their data and from here, the PI and/or student might decide which data could/should be copied into a more collaborative group where additional members would also be able to view the data.
- For an institutional repository type structure where data are being archived, but not necessarily open for general viewing.

Read-only This group is the intermediate option that 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 some annotations on the other group members data.
- *Group member* can see who other members are and view their data, but he 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. Other members can view, while the group owners can annotate and/or add Regions of Interest (ROIs) to the other member's images.
- 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 is the most collaborative group:

- *Group member* can view other members, their data and can make annotations on those other members' data.

Potential Use-Cases:

- This could be used by a group of scientists working together with data for a publication.

See also:

OMERO.insight Permissions update in OMERO 4.4[3] Movie describing the permissions update under OMERO.insight in OMERO 4.4

Web Permissions update in OMERO 4.4[4] Movie describing the permissions update under Web in OMERO 4.4

10.1.4 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*. Once links have been created in the database under *Read-only* or *Read-annotate* permissions, these cannot be severed. However, it is possible to 'promote' a *Private* group to be a *Read-only* or *Read-annotate* group.
- It is possible to toggle permissions of a group between the two collaborative *Read-only* and *Read-annotate* groups.

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.

10.1.5 Permissions on your and other users' data

What can you do with your data?

All OMERO users in all groups can perform all actions to their own 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 comment).
- De-annotate (remove annotations that you have added).
- Use Regions of Interest (ROIs) (add, import, edit, delete, save and analyze with 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[5] or through the forums[6].

10.1.6 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.

Administrator

Action	*Private*	*Read-only*	*Read-annotate*
View	Y	Y	Y
Annotate	N	Y	Y
Delete	Y	Y	Y
Edit	Y	Y	Y
Move between groups	Y	Y	Y
Remove annotations	Y	Y	Y
Mix data	N	Y	Y

Group owner

[5] http://www.openmicroscopy.org/site/community/mailing-lists
[6] http://www.openmicroscopy.org/community/

Action	*Private*	*Read-only*	*Read-annotate*
View	Y	Y	Y
Annotate	N	Y	Y
Delete	Y	Y	Y
Edit	Y	Y	Y
Move between groups	N	N	N
Remove annotations	Y	Y	Y
Mix data	N	Y	Y

Group member

Action	*Private*	*Read-only*	*Read-annotate*
View	N	Y	Y
Annotate	N	N	Y
Delete	N	N	N
Edit	N	N	N
Move between groups	N	N	N
Remove annotations	N	N	N
Mix data	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).

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 the admin has the right to move other users' data between groups.

Note: The admin does not have to be member of 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.

10.1.7 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

10.2 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.

10.2.1 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 will currently function on the following systems:
  - Linux with kernel 2.6.13 and higher.
  - Mac OS 10.5 and above.
  - Windows XP and Windows Server 2003.
- In addition some platforms require further Python packages to be available:
  - Linux servers 4.2.x or earlier require Pyinotify 0.7.x[7] or Pyinotify 0.8.x[8]. Some Linux distributions have one or other of these packages pre-installed. Some distributions of Pyinotify versions 0.8.6 and 0.8.7 are not compatible with Python 2.4. If your system runs Python 2.4 Pyinotify 0.8.5 or lower is recommended.
  - Linux systems running Python 2.4 also requires ctypes (bundled with Python 2.5+) which is available from http://python.net/crew/theller/ctypes/
  - 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/2.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.

[7] http://pyinotify.sourceforge.net/
[8] http://trac.dbzteam.org/pyinotify/

10.2.2 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.

Acquisition systems can then be configured to drop a user's images into a given DropBox.

Note: The DropBox system is designed for images filesets to be copied in at normal acquisition rates. Copying numbers of files en masse may result in files failing to import.

10.2.3 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.

10.2.4 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. 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

  The importUsers is either `default` for the standard DropBox configuration or a list of OMERO user names. The default 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 *The Command Line Import*.

  ```
  <property name="omero.fs.importArgs"  value=""/>
  ```

Example

Here's 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="--report;--report --email zak@example.com"/>
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

10.3 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[9]'s IceGrid framework, OMERO.grid provides an administration GUI, distributed background processing, log handling, and several other features.

10.3.1 Getting started

Requirements

OMERO.grid is the basis of the regular OMERO installation. If you have followed the instructions under *Unix* or *Windows*, then you will have everything you need to start working with OMERO.grid.

IceGrid Tools

If you would like to exploring your IceGrid configuration, use

```
bin/omero admin ice
```

It provides full access to the `icegridadmin` console described in the ZeroC[10] manual. Specific commands can also be executed:

```
bin/omero admin ice help
bin/omero admin ice application list
bin/omero admin ice application describe etc/grid/default.xml
bin/omero admin ice server list
```

Further, by running `java -jar ice-gridgui.jar` the GUI provided ZeroC[11] can be used to administer OMERO.grid. This jar is provided in the OMERO source code under lib/repository.

See also:

Administrative Utilities[12] Chapter of the ZeroC[13] manual about administrative clients

OMERO.grid on Windows

Unlike all other supported platforms, the `bin\omero` script and OMERO.grid are not directly responsible for starting and stopping the *OMERO.blitz* server and other processes. Instead, that job is delegated to the native Windows service system. A brief explanation is provided below.

[9] http://www.zeroc.com
[10] http://www.zeroc.com
[11] http://www.zeroc.com
[13] http://www.zeroc.com

```
bin\omero admin start
bin\omero admin deploy
...
bin\omero admin stop
```

The first command installs OMERO.grid as a Windows service with the name `OMERO.master` and starts it. Any further calls to `omero admin start` will fail since the application is already installed as a Windows service. Similarly, `omero admin stop` first stops the service, and then removes it. See *OMERO.server Windows Service*.

Further interactions with the Windows service can take place via `sc.exe`.

```
sc start OMERO.master
sc stop OMERO.master
sc delete OMERO.master
sc query OMERO.master
```

More information on the permissions necessary for the service, changing the user it runs as, etc. are available under `WINDOWS_SERVICE.txt` under your Ice installation, most likely `C:\Ice-3.3.1\WINDOWS_SERVICE.txt`

10.3.2 How it works

IceGrid[14] 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[15] . 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 three nodes – "master", "node1", "node2". 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 "servers" are started on which "nodes". For example the default[16] 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 master.cfg[17] 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 default.xml:

```
bin/omero admin start
```

The deploy command looks for any changes to the defined descriptor and restarts only those servers which have modifications:

```
bin/omero admin deploy
```

Both `start` and `deploy` can optionally take a path to an application descriptor which must be passed on every invocation:

[14] http://zeroc.com/
[15] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/etc/grid
[16] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/default.xml
[17] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/master.cfg

```
bin/omero admin deploy etc/grid/my-site.xml target1 target2
```

Two other nodes, then, each provide a single processor, "Processor1" and "Processor2". These are started via:

```
bin/omero node start NAME
```

at which point they connect to the registry to announce their presence. Now, jobs can be run on any of the 3 processors. 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 stop` on that particular node.

Modifying deployments

The most common change that you will want to make to your application descriptor is to add another processor. Take a look at etc/grid/default.xml[18]. There are two nodes which are defined: **node1** and **node2**. To add another processing node, simply copy the node element:

```
<node name="node1">
  <server-instance template="ProcessorTemplate" index="1"/>
</node>
```

and change the node name and the index number.

```
<node name="MyNewNode">
  <server-instance template="ProcessorTemplate" index="3"/>
</node>
```

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:

```
bin/omero node MyNewNode start
```

or with the environment variable `OMERO_NODE`:

```
OMERO_NODE=MyNewNode bin/omero node start
```

For more information on using scripts, see the *OMERO.scripts advanced topics*.

10.3.3 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.

[18] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/default.xml

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

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 etc/master.cfg[19]:

```
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 admin 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. (We are currently looking into providing a root- or admin-only permissions verifier for public use.)

See Section 39.11.2 Access Control[20] 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.

[19] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/master.cfg
[20] http://zeroc.com/doc/Ice-3.2.1/manual/IceGrid.40.11.html#108430

Absolute paths

Except under Windows, 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 who ever starts the node have access to that directory. Therefore, as long as you control the boxes which can attached to your endpoints (see *Firewall*), then you can be relatively certain that no tampering can occur with the installed binaries.

10.3.4 Technical information and other tips

Processes

It is important to understand just what processes will be running on your servers. When you run `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 `etc/grid/default.xml` the section which defines the main *OMERO.blitz* server includes:

```
<server id="Blitz-${index}" exe="${exe}" activation="always">
  <target name="debug">
    <option>-Xdebug</option>
    <option>-Xrunjdwp:server=y,transport=dt_socket,address=${port},suspend=n</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:

```
bin/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[21]) and configuration files (e.g. ice.config[22]). 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.

[21] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/templates.xml
[22] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/ice.config

Shortcuts

If the `bin/omero` script is copied or symlinked to another name, then the script will separate the name on hyphens and execute `bin/omero` with the second and later parts **prepended** to the argument list.

For example,

```
ln -s bin/omero bin/omero-admin
bin/omero-admin start
```

works identically to:

```
bin/omero admin start
```

Symbolic linking

Shortcuts allow the `bin/omero` script to function as a 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 unless `OMERO_HOME` is set.

For example,

```
ln -s $OMERO_PREFIX/bin/omero /usr/local/bin/omero
omero-admin start
```

The same works for putting `bin/omero` on your path:

```
PATH=$OMERO_PREFIX/bin:$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[23]. (This holds only for Unix-based platforms including MacOsX. See *OMERO.grid on Windows* for information on changing the server user under Windows.)

See also:

OMERO sessions

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

[23] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/templates.xml

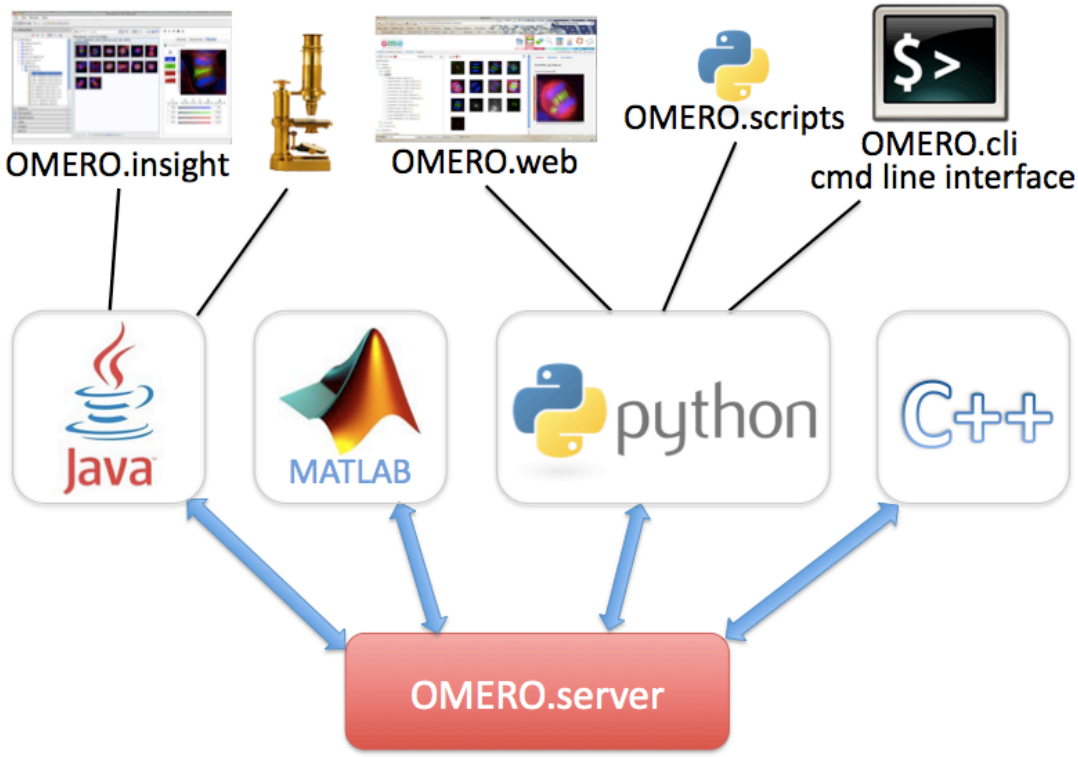
Part III

Developer Documentation

Warning: With the release of OMERO 5.0, the 4.4.x line has now entered maintenance mode. We will continue to support this version throughout 2014 but it will only be updated for major bug fixes, so you may wish to work against the new 5.0 version[a] instead.

[a] http://www.openmicroscopy.org/site/support/omero5/developers/

The following documentation is for developers wishing to write OMERO client code or extend the OMERO server. Instructions on downloading[24], 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 *OMERO Command Line Interface*, 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[25], with contributions from Glencoe Software, Inc.[26] OMERO is released under the GNU General Public License (GPL)[27] with commercial licenses and customization available from Glencoe Software, Inc.[28].

For help with any aspect of OMERO, see details of our forums and mailing lists[29].

[24] http://downloads.openmicroscopy.org/latest/omero4/
[25] http://www.openmicroscopy.org/site
[26] http://www.glencoesoftware.com/
[27] http://www.gnu.org/copyleft/gpl.html
[28] http://www.glencoesoftware.com/
[29] http://www.openmicroscopy.org/site/community/

CHAPTER

ELEVEN

INTRODUCTION TO OMERO

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

11.1 Installing OMERO from source

This section describes how to check out and build the OMERO source code on your local machine. To install the dependencies required to run the OMERO.server on Linux or Mac OS X, take a look at the *OMERO.server Linux installation walk-through* or the *OMERO.server Mac OS X installation walk-through with Homebrew*.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

11.1.1 Checking out the source code

This section is primarily designed for the core OME developers who want to check out the main code base using git.

Code locations

OME code is stored in multiple git repositories, each of which is available from several locations.

OMERO

The main repository, known as ome.git, is available from:

- https://github.com/openmicroscopy/openmicroscopy
- git://openmicroscopy.org/ome.git

Bio-Formats

The Bio-Formats repository is available from:

- https://github.com/openmicroscopy/bioformats
- git://openmicroscopy.org/bioformats.git

Most of these repositories are read-only locations which are kept in sync for public consumption. There is a third centrally-available location but this is not public:

- ssh://git.openmicroscopy.org/home/git/team.git

`team.git` is useful for team-internal branches which are not yet ready for public consumption. Changes that are not ready to be reviewed by the public can be exchanged via team. This allows you to internally collaborate on a branch or simply to back-up your work.

After that, each member of the GitHub openmicroscopy organization (https://github.com/openmicroscopy), as well as anyone else who has clicked the "Fork" button, will have their own repository. These are listed here:

- https://github.com/openmicroscopy/openmicroscopy/network/members

Installing git

In general, see the Git downloads page[1] for installation options.

Linux

Most flavors of Linux have git available through the package manager. For example, on Debian/Ubuntu:

```
sudo apt-get install git
```

Mac OS X

You can install git using Homebrew[2]:

```
brew install git
```

Or you can use the binary installer[3].

Windows

We recommend using either msysGit[4] for a basic git installation, or Cygwin[5] for a full-featured Unix-style environment that includes git. You can also use TortoiseGit[6] for git shell integration. You may also want to consider installing VirtualBox[7] with a Linux guest OS to make your life easier. Lastly, when using git on Windows, please be aware of the CRLF conversion issue[8].

Git configuration

If you are looking to get started as quickly as possible, the minimum you will need is to have git installed (see next section) and then:

```
git config --global user.name "Full name"
git config --global user.email YOUR_EMAIL
git clone --recursive https://github.com/openmicroscopy/openmicroscopy
cd openmicroscopy
```

You will not be able to push back to this repository, but you will at least have something you can start looking at.

Git provides a number of options which can make working with it considerably more pleasant. These configuration options are available either globally in `$HOME/.gitconfig` or in the `.git` directory of your repository. The file is in INI-format, but can also be modified using the `git config` command, as illustrated in the examples following.

The most important thing is to update your 'global' credentials that are used in your commits. These values are saved in `~/.gitconfig`:

[1] http://git-scm.com/download
[2] https://github.com/mxcl/homebrew/
[3] http://git-scm.com/download
[4] http://code.google.com/p/msysgit
[5] http://www.cygwin.com/
[6] http://code.google.com/p/tortoisegit/
[7] https://www.virtualbox.org/
[8] http://help.github.com/articles/dealing-with-line-endings

```
git config --global user.name "Full name"
git config --global user.email YOUR_EMAIL
```

If you have a PGP key for signing commits and tags, you may want to add it as well:

```
git config --global user.signingkey YOUR_PGP_KEY_ID
```

Color and display options make log and diff output much more friendly:

```
git config --global color.ui true
git config --global color.diff auto
git config --global color.graph auto
git config --global color.status auto
git config --global color.branch auto

git config --global core.ui always
git config --global core.editor mate_wait
```

Aliases provide a way to make shortcuts for longer git commands. One that is often used among the OME team is `graph`:

```
git config --global alias.graph "log --date-order --graph --decorate --oneline"
```

See Helpful command aliases[9] for more examples.

Cloning the source code

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/openmicroscopy/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/openmicroscopy/openmicroscopy.git
```

The natural workflow when using GitHub is not just to download someone else's repository, but also to create a personal working copy. Go to the repository page at https://github.com/openmicroscopy/openmicroscopy and click on "Fork". This will create a copy of the repository in your own personal space:

```
https://github.com/YOURNAME/openmicroscopy
```

which can be added to your local repository as another remote:

[9] http://gitready.com/intermediate/2009/02/06/helpful-command-aliases.html

```
git remote add gh git@github.com:YOURNAME/openmicroscopy.git
```

Note: For the SSH transport to work, you will need to follow some of the instructions under https://github.com/account/ssh

Depending on which repository you cloned first, either origin/develop or gh/develop will be the "develop" branch of your own fork of openmicroscopy/openmicroscopy. The example below assumes that "gh" is your own personal GitHub repository, and "origin" is the official openmicroscopy repository.

You may even want to remove the "develop" branch from your fork since all branching should happen from the official develop branch. If you'd prefer to keep a copy of "develop" in "gh", that is fine, but you may then need to keep your develop up-to-date with the official develop:

```
git checkout develop
git reset --hard origin/develop   # Warning: This will delete any unsaved changes and commits to develo
git push -f gh develop            # Warning: This will replace gh/develop with the official version remo
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

11.1.2 Build System

Overview

The page goes into details about how the build system is configured.

Creating binary distribution

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.

Note: By default, *OMERO C++ language bindings* is not built. Use `build-all` for that.

OMERO_SOURCE_PREFIX	OMERO_SOURCE_PREFIX/dist	Comments
components/blitz/target/blitz.jar	lib/server	Primary Ice servants
components/blitz/target/server.jar	lib/server	Primary server logic
components/tools/OmeroCpp/lib*	lib/	Native shared libraries
components/tools/OmeroPy/build/lib	lib/python	Python 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
<javadoc/>	docs/api	(Optional) Javadocs produced with "java omero javadoc"

These files are then zipped to OMERO.server-<version>.zip via "java omero release-zip"

Jenkins

The OME project currently uses Jenkins[10] as a continuous integration server available here[11], so many binary packages can be downloaded without compiling them yourself. OMERO.server is built by the "OMERO" job[12].

[10] http://jenkins-ci.org
[11] http://ci.openmicroscopy.org/
[12] http://ci.openmicroscopy.org/job/OMERO-trunk/

Hudson checks for git changes every 15 minutes and executes:

```
(cd docs/hudson; python launcher.py)
```

which invokes the "build-all", "javadoc" "findbugs", "coverage", and "release-zip" targets.

The Javadocs are always made available here[13] as well as several build metrics.

Build tools

OMERO mostly uses an ant[14]-based build with dependency management provided by Ivy[15]. *Native code* is built using SCons[16] 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 driver
|
|-- build.py .......................... Python wrapper to handle OS-specific configuration
|
|-- omero.class ....................... Self-contained Ant launcher
|
|--etc/ ............................... All configuration
|   |-- grid/* ........................ Deployment files
|   |-- ivysettings.xml
|   |-- hibernate.properties
|   |-- local.properties.example
|   |-- log4j.xml
|   |-- omero.properties
|   \-- profiles
|
|-- examples .......................... User examples
|
\components
  |
  |
  |--<component-name> ................... Each component has this same basic structure.
  |    |-- build.xml                     Main scripts
  |    |-- ivy.xml                       Jar dependencies
  |    |-- test.xml                      Test dependencies
  |    |-- src                           Source code
  |    |-- resources                     Other files of interest
  |    |-- test                          Test source code and test resources
  |    \-- target                        Output of build (deleted on clean)
  |
  |-- model ............................. The model component is special in that in produces
  |                                       a jar specific to your database choice: model-psql.jar
  |                                       The generated 'ome.model.*' files contain Hibernate
  |                                       annotations for object-relational mapping.
  |
  |-- blitz ............................. The blitz component also performs code generation
  |    |                                  producing artifacts for Java, Python, and C++.
  |    |
  |    \ blitz_tools.py ................. OMERO-specific SCons Environment definition
```

[13] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/
[14] http://ant.apache.org
[15] http://ant.apache.org/ivy
[16] http://scons.org

```
|                                    and other build tools.
|
|--tools ............................ Other server-components with special build needs.
|    |--build.xml
|    \--<tool-name>
|         |--build.xml
|         `--ivy.xml
|
\--antlib ........................... Special component which is not built, but referenced by the b
    |
    \--resources
        |--global.xml
        |--directories.xml
        |--lifecycle.xml
        \--depdendencies.xml
```

Note: User examples are explained under *Working with OMERO*

Each of the components can also be built directly. For example,

```
./build.py -f components/server/build.xml
```

Code generation

Unfortunately, just the above snapshot of the code repository omits some of the most important code. Many MB of source code is generated both by our own DSLTask[17] as well as by the Ice[18] `slice2java`, `slice2cpp`, and `slice2py` code generators. These take an intermediate representation of the OME-Model[19] 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".

OmeroTools

Similarly, 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 components/tools[20] are a bit more complex. Each tools component installs its artifacts to the tools/target directory which is copied **on top of** the OMERO_HOME/dist top-level distribution directory. Current tools include:

	Ant-based builds	*Ice-based builds*	*Scons-based builds*
OMERO C++ language bindings			X
OMERO.web framework	X		
OMERO.fs		X	
OMERO Python language bindings		X	
LicenseService	X		X

Ant-based builds Some of the tools also contain Java code which imports files from `antlib/resources` and then proceeds like the other regular components.

Ice-based builds An Ice-based build requires further invocations of `slice2*` code generation. Currently this

Scons-based builds Builds which have C++ targets are based generally on Scons[21]. See *OMERO C++ language bindings* for more information.

[17] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/dsl
[18] http://www.zeroc.com
[19] http://www.openmicroscopy.org/site/support/ome-model/ome-xml/
[20] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/tools
[21] http://www.scons.org

Comments on Ivy

- Resolvers are key to how Ivy functions. Currently, the default resolver is called "omero-resolver" and simply looks in our repository (`./lib/repository`) for the jars which were downloaded from git. Multi-resolvers can be defined (as granular as for an individual jar) in order to pick up the latest version of whatever library from HTTP, SSH, or from the local file system.
- OMERO_HOME/lib/cache : in order to determine the transitive closure of all dependencies, Ivy "resolves" each ivy.xml and stores the resolved ivy.xml in its cache (in our build, `./lib/cache`) to speed up other processes. However, when changing the Ivy configuration (`./etc/ivyconf.xml`) or version number (`etc/omero.properties->omero.version`) the cache can become stale. This should not happen, but currently does. It may be beneficial for the time being to call `ant clean` from the top-level build which will delete the cache.

Comments on build.py

`./build.py` is a complete replacement for your local ant install. In many cases on and on most OS, 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

11.2 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*.

11.2.1 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*.

11.2.2 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[22].

Regular builds of the server are performed by Jenkins[23] (formerly known as hudson) and all results are available here here[24], including generated javadocs[25].

11.2.3 Environment variables

In addition to the install instructions, you might find it useful to set the following variables:

- OMERO_HOME: should be set to the directory containing the OMERO distribution or build

```
# E.g. If you built the server yourself
export OMERO_PREFIX=~/Desktop/OMERO/dist
# Or you downloaded a release package
export OMERO_PREFIX=~/Desktop/OMERO.server-Beta-4.2.2
```

- Add the /bin/ directory to your PATH - allows you to call the 'omero' command from anywhere

```
export PATH=$PATH:$OMERO_PREFIX/bin/
```

- For Python developers, set your PYTHONPATH as follows

```
export PYTHONPATH=$PYTHONPATH:$OMERO_PREFIX/lib/python/
```

Now checkout the CLI.

```
$ omero -h
```

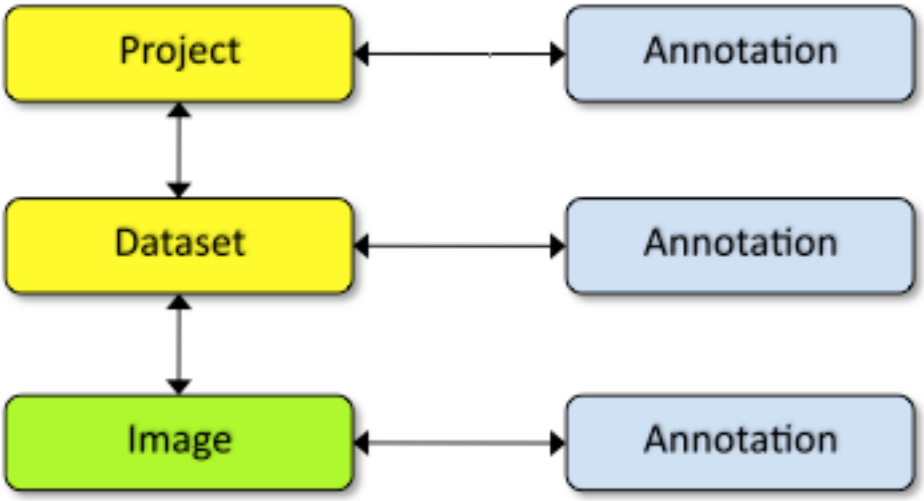
11.2.4 Database access

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[26] is a free, cross platform GUI tool for working with PostgreSQL



11.2.5 OMERO model

You can browse the OMERO model in a number of ways, one of which is by looking at the database itself (see above). Another is via the on-line OMERO model[27] docs.

However, due to the complexity of the OMERO model, it is helpful to have some starting points (follow links below to the docs themselves).

Note: Figures to the right show highly simplified outline of various model objects.

Projects, datasets and images

Projects[28] and Datasets[29] are many-to-many containers for Images[30] (linked by ProjectDatasetLinks[31] and DatasetImageLinks[32] respectively).

Projects, Datasets, Images and a number of other entities can be linked to Annotations (abstract superclass)[33] via specific links (ProjectAnnotationLink[34], DatasetAnnotationLink[35] etc). Annotation subclasses such as CommentAnnotation[36], FileAnnotation[37] etc. are stored in a single database table in OMERO (all Annotations have unique ID).

Images

Images in OMERO are made up many entities. These include core image components such as Pixels[38] and Channels[39], as well as a large number of additional metadata objects such as Instrument (microscope), Objective, Filters, Light Sources, and Detectors.

[22] http://downloads.openmicroscopy.org/latest/omero4/
[23] http://jenkins-ci.org
[24] http://ci.openmicroscopy.org/job/OMERO-trunk/
[25] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/
[26] http://www.pgadmin.org/
[27] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model.html
[28] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/Project.html
[29] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/Dataset.html
[30] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/Image.html
[31] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/ProjectDatasetLink.html
[32] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/DatasetImageLink.html
[33] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/Annotation.html
[34] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/ProjectAnnotationLink.html
[35] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/DatasetAnnotationLink.html
[36] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/CommentAnnotation.html
[37] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/FileAnnotation.html
[38] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/Pixels.html
[39] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model/Channel.html

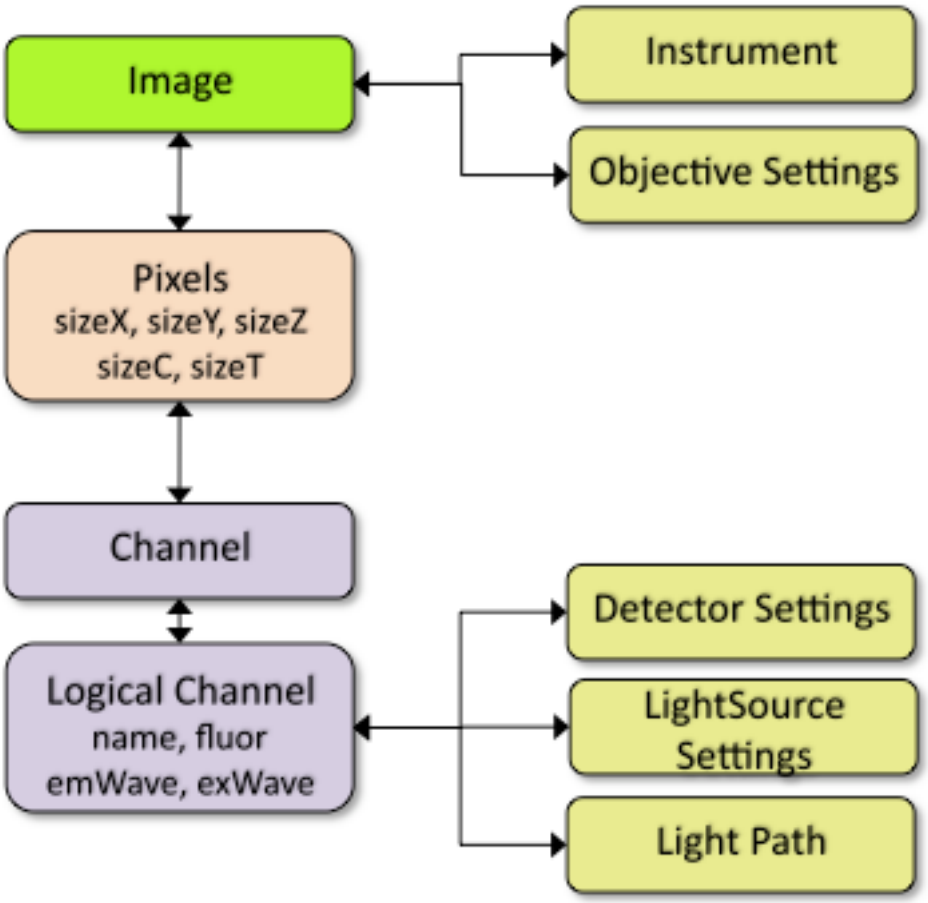


Image pixels data is stored as a single file, numbered by Pixels ID in the OMERO repository under /OMERO/Pixels/ and can be accessed using the Raw Pixels Store.

Note: Some features of the model are due to historical changes and are not exercised in practice. For example, an Image can have multiple sets of Pixels although only 1 is typically used. Also, Logical Channel and Channel can be considered a single entity (as they are now in the OME-model).

11.2.6 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 2 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.

11.2.7 Examples

HQL examples

HQL is used for Query Service queries (see above). Some examples, coupled with a knowledge of the OMERO model 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 PixelsType as p where p.value='%s'" % pType, None)

# better to do
params = omero.sys.Parameters()
params.map = {"pType": rstring(pType)}
queryService.findByQuery("from PixelsType 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

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
 id |       name                        |          path                                                 | owner
----+-----------------------------------+---------------------------------------------------------------+------
 56 | GFP-FRAP.cpe.xml                  | /Users/will/omero/editor/GFP-FRAP.cpe.xml                     |

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 | par
----+-------------+---------+-------+-------------+-------------+----------+----------+-----------+----
 14 |        -103 |         |   110 |         891 |             |      208 |      207 |       891 |
 17 |        -103 |         |   113 |         926 |             |      208 |      207 |       926 |
(2 rows)

omero=# select id, name, path, owner_id, group_id, ome_perms(permissions) from originalfile where id in
 id  |       name        |                              path                                | owner_id | gr
-----+-------------------+------------------------------------------------------------------+----------+---
 113 | stdout            | /Users/will/omero/tmp/omero_will/75270/processuLq8fd.dir/out |      207 |
 110 | imagesFromRois.py | ScriptName061ea79c-f98c-447b-b720-d17003d6a72f                |        0 |
(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 ;
```

bin/omero hql

You can use the omero `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.

```
bin/omero hql -q --limit=10 "select name from OriginalFile where id=4106"
bin/omero hql -q --limit=10 "select id, textValue, file from Annotation a order by a.id desc"
bin/omero hql -q --limit=10 "select id, textValue from TagAnnotation a order by a.id desc"
bin/omero hql -q --limit=100 "select id, owner.id, started, userAgent from Session where closed is null
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

11.3 Contributing to OMERO

Note: This section of the documentation has been updated and is now hosted at http://www.openmicroscopy.org/site/support/contributing/ or available as a pdf[40].

[40] http://www.openmicroscopy.org/site/support/contributing/OME-Contributing-Developer.pdf

CHAPTER

TWELVE

USING THE OMERO API

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

12.1 OMERO Python language bindings

MOVIE: introduction to Blitz Gateway[1]

In addition to the auto-generated Python libraries of the core *OMERO Application Programming Interface*, we have developed a more user-friendly Python module 'Blitz Gateway' that facilitates several aspects of working with the Python API, such as connection handling, object graph traversal and lazy loading.

This page first gives you a large number of code samples to get you started. Below these we describe a bit more about *using the Blitz Gateway*.

The Python libraries are part of the server build and can be found under OMERO_HOME/lib/python. These include the core omero.model objects and services as well as the Blitz Gateway code (at OMERO_HOME/lib/python/omero/gateway/__init__.py).

To use OmeroPy, you will need to download the libraries (E.g. as part of the server package) and setup your PYTHONPATH to include them:

```
export OMERO_PREFIX=~/Desktop/OMERO.server-4.4.12-ice3x-byy       # for example
export PYTHONPATH=$OMERO_PREFIX/lib/python
```

You will also need Ice libraries 3.3.x or 3.4.x as described in the *OMERO.server installation* and an OMERO server to connect to, which must be the same major version, i.e. 4.4.x.

All the code examples below can be found at examples/Training/python[2].

Start by downloading the first example below: examples/Training/python/Connect_To_OMERO.py[3] and edit the USERNAME and PASSWORD variables according to your log-in.

Then you can run the example:

```
$ python Connect_To_OMERO.py
```

If all goes well, you should be connected to your OMERO server and see some details of your session printed out.

All the following code examples can be downloaded and run in the same way, and they will use the USERNAME and PASSWORD from the file you just edited. However, you will need to edit some other parameters, usually IDs from Projects, Datasets, Images etc. You can use the OMERO.insight or OMERO.web client to choose suitable data IDs before editing and running the code samples.

[1] http://cvs.openmicroscopy.org.uk/snapshots/movies/omero-4-3/mov/BlitzGatewayIntro-4.3.mov
[2] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/examples/Training/python
[3] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/Training/python/Connect_To_OMERO.py

12.1.1 Code samples

Connect to OMERO

```
# These values will be imported by all the other training scripts.
HOST = 'localhost'
PORT = 4064
USERNAME = 'username'
PASSWORD = 'passwd'


from omero.gateway import BlitzGateway


if __name__ == '__main__':
    """
    NB: This block is only run when calling this file directly
    and not when imported.
    """


# Connect to the Python Blitz Gateway
# ===========================================================
# Make a simple connection to OMERO, printing details of the
# connection. See OmeroPy/Gateway for more info
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
connected = conn.connect()


# Check if you are connected.
# ===========================================================
if not connected:
    import sys
    sys.stderr.write("Error: Connection not available, please check your user name and password.\n")
    sys.exit(1)


# Using secure connection.
# ===========================================================
# By default, once we have logged in, data transfer is not encrypted (faster)
# To use a secure connection, call setSecure(True):


# conn.setSecure(True)          # <--------- Uncomment this


# 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()
```

```
print "Member of:"
for g in conn.getGroupsMemberOf():
    print "   ID:", g.getName(), " Name:", g.getId()
group = conn.getGroupFromContext()
print "Current group: ", group.getName()



print "Other Members of current group:"
for exp in conn.listColleagues():
    print "   ID:", exp.getId(), exp.getOmeName(), " Name:", exp.getFullName()



print "Owner of:"
for g in conn.listOwnedGroups():
    print "   ID:", g.getName(), " Name:", g.getId()



# The 'context' of our current session
ctx = conn.getEventContext()
# print ctx     # for more info



# Close connection:
# ====================================================================
# When you are done, close the session to free up server resources.
conn._closeSession()
```

Read data

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
```

- **Configuration**

```
imageId = 1
datasetId = 2
plateId = -1        # Don't need to set this



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:**

```
# The only_owned=True parameter limits the Projects which are returned.
# If the parameter is omitted or the value is False, then all Projects
# visible in the current group are returned.
print "\nList Projects:"
print "=" * 50
my_expId = conn.getUser().getId()
for project in conn.listProjects(my_expId):
    print_obj(project)
    for dataset in project.listChildren():
        print_obj(dataset, 2)
        for image in dataset.listChildren():
            print_obj(image, 4)
```

- **Retrieve the datasets owned by the user currently logged in:**

```
# Here we create an omero.sys.ParametersI instance which we
# can use to filter the results that are returned. If we did
# not pass the params argument to getObjects, then all Datasets
# in the current group would be returned.
print "\nList Datasets:"
print "=" * 50



params = omero.sys.ParametersI()
params.exp(conn.getUser().getId())  # only show current user's Datasets



datasets = conn.getObjects("Dataset", params=params)
for dataset in datasets:
    print_obj(dataset)
```

- **Retrieve the images contained in a dataset:**

```
print "\nDataset:%s" % datasetId
print "=" * 50
dataset = conn.getObject("Dataset", datasetId)
print "\nImages in Dataset:", dataset.getName()
for image in dataset.listChildren():
    print_obj(image)
```

- **Retrieve an image by Image ID:**

```
image = conn.getObject("Image", imageId)
print "\nImage:%s" % imageId
print "=" * 50
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()
# render the first timepoint, mid Z section
z = image.getSizeZ() / 2
t = 0
renderedImage = image.renderImage(z, t)
#renderedImage.show()                # popup (use for debug only)
#renderedImage.save("test.jpg")      # save in the current folder
```

- **Retrieve Screening data:**

```
print "\nList Screens:"
print "=" * 50
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:**

```
if plateId >= 0:
    print "\nPlate:%s" % plateId
    print "=" * 50
    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 xrange(0, index):
            print "    Image: ", \
                    well.getImage(index).getName(),\
                    well.getImage(index).getId()
```

- **Close connection:**

```
# When you are done, close the session to free up server resources.
conn._closeSession()
```

Groups and permissions

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
```

- **Configuration**

```
imageId = 1
```

- **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'}  # Not exposed in 4.4.0 clients
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 *Permissions overview*.

```
projects = conn.listProjects()      # may include other users' data
for p in projects:
    print p.getName(), "Owner: ", p.getDetails().getOwner().getFullName()
```

```
image = conn.getObject("Image", imageId)     # Will return None if Image is not in current group
print "Image: ", image
```

- **In OMERO-4.4, we added '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.details.group.id.val    # access groupId without loading group
```

- **To query only a single group (not necessarily your 'current' group)**

```
groupId = image.details.group.id.val
conn.SERVICE_OPTS.setOmeroGroup(groupId)   # This is how we 'switch group' in webclient
projects = conn.listProjects()
image = conn.getObject("Image", imageId)
print "Image: ", image,
```

- **Close connection:**

```
# When you are done, close the session to free up server resources.
conn._closeSession()
```

Raw data access

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
```

- **Configuration**

```
imageId = 27544
```

- **Retrieve a given plane**

```
# Use the pixelswrapper to retrieve the plane as
# a 2D numpy array. See [http://www.scipy.org/Tentative_NumPy_Tutorial]
#
# Numpy array can be used for various analysis routines
#
image = conn.getObject("Image", imageId)
sizeZ = image.getSizeZ()
sizeC = image.getSizeC()
sizeT = 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))....etc... ]
zctList = [(z, c, t, tile) for z in range(sizeZ)]
print "\nZ stack of tiles:"
planes = pixels.getTiles(zctList)
for i, p in enumerate(planes):
    print "Tile:", zctList[i], " min:", p.min(),\
            " max:", p.max(), " sum:", p.sum()
```

- **Retrieve a given hypercube**

```
zctList = []
for z in range(sizeZ / 2, sizeZ):     # get the top half of the Z-stack
    for c in range(sizeC):          # all channels
        for t in range(sizeT):      # all time-points
            zctList.append((z, c, t))
print "\nHyper stack of planes:"
planes = pixels.getPlanes(zctList)
for i, p in enumerate(planes):
    print "plane zct:", zctList[i], " min:", p.min(), " max:", p.max()
```

- **Close connection:**

```
# When you are done, close the session to free up server resources.
conn._closeSession()
```

Write data

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
```

- **Configuration**

```
projectId = 2
#Specify a local file. E.g. could be result of some analysis
fileToUpload = "README.txt"    # This file should already exist
```

- **Create a new Dataset**

```
datasetObj = omero.model.DatasetI()
datasetObj.setName(rstring("New Dataset"))
datasetObj = conn.getUpdateService().saveAndReturnObject(datasetObj)
datasetId = datasetObj.getId().getValue()
print "New dataset, Id:", datasetId
```

- **Link to Project**

```
project = conn.getObject("Project", projectId)
if project is None:
    import sys
    sys.stderr.write("Error: Object does not exist.\n")
    sys.exit(1)
link = omero.model.ProjectDatasetLinkI()
link.setParent(omero.model.ProjectI(project.getId(), False))
link.setChild(datasetObj)
conn.getUpdateService().saveObject(link)
```

- **How to create a file annotation and link to a Dataset**

```
dataset = conn.getObject("Dataset", datasetId)
# create the original file and file annotation (uploads the file etc.)
namespace = "imperial.training.demo"
print "\nCreating an OriginalFile and FileAnnotation"
fileAnn = conn.createFileAnnfromLocalFile(fileToUpload, mimetype="text/plain", ns=namespace, desc=None)
print "Attaching FileAnnotation to Dataset: ", "File ID:", fileAnn.getId(), ",", fileAnn.getFile().getNam
dataset.linkAnnotation(fileAnn)      # 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.
print "\nAnnotations on Dataset:", dataset.getName()
for ann in dataset.listAnnotations():
    if isinstance(ann, omero.gateway.FileAnnotationWrapper):
        print "File ID:", ann.getFile().getId(), ann.getFile().getName(), "Size:", ann.getFile().getSiz



file_path = os.path.join(path, ann.getFile().getName())
```

```
f = open(str(file_path), 'w')
print "\nDownloading file to", file_path, "..."
try:
    for chunk in ann.getFileInChunks():
        f.write(chunk)
finally:
    f.close()
    print "File downloaded!"
```

- **Load all the file annotations with a given namespace**

```
nsToInclude = [namespace]
nsToExclude = []
metadataService = conn.getMetadataService()
annotations = metadataService.loadSpecifiedAnnotations('omero.model.FileAnnotation', nsToInclude, nsToEx
for ann in annotations:
    print ann.getId().getValue(), ann.file.name.val
```

- **Get first annotation with specified namespace**

```
ann = dataset.getAnnotation(namespace)
print "Found Annotation with namespace: ", ann.getNs()
```

- **Close connection:**

```
# When you are done, close the session to free up server resources.
conn._closeSession()
```

OMERO tables

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
```

- **Configuration**

```
datasetId = 33
```

- **Create a name for the Original File (should be unique)**

```
from random import random
tablename = "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.**

```
repositoryId = 1
table = conn.c.sf.sharedResources().newTable(repositoryId, tablename)
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)
table.close()           # when we are done, close.
```

- **Get the table as an original file…**

```
orig_file = table.getOriginalFile()
orig_file_id = orig_file.id.val
# ...so you can attach this data to an object. E.g. Dataset
fileAnn = omero.model.FileAnnotationI()
fileAnn.setFile(omero.model.OriginalFileI(orig_file_id, False))      # use unloaded OriginalFileI
fileAnn = conn.getUpdateService().saveAndReturnObject(fileAnn)
link = omero.model.DatasetAnnotationLinkI()
link.setParent(omero.model.DatasetI(datasetId, False))
link.setChild(omero.model.FileAnnotationI(fileAnn.id.val, False))
conn.getUpdateService().saveAndReturnObject(link)
```

- **Table API**

  See also:

  javadoc[4]

```
openTable = conn.c.sf.sharedResources().openTable(orig_file)

print "Table Columns:"
for col in openTable.getHeaders():
    print "    ", col.name

rowCount = openTable.getNumberOfRows()
print "Row count:", rowCount
```

- **Get data from every column of the specified rows**

```
rowNumbers = [3, 5, 7]
print "\nGet All Data for rows: ", rowNumbers
data = openTable.readCoordinates(range(rowCount))
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**

[4] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/grid/Table.html

```
colNumbers = [1]
start = 3
stop = 7
print "\nGet Data for cols: ", colNumbers,\
        " and between rows: ", start, "-", stop



data = openTable.read(colNumbers, 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**

```
queryRows = openTable.getWhereList("(Uid > 2) & (Uid <= 8)",\
        variables={}, start=0, stop=rowCount, step=0)
data = openTable.readCoordinates(queryRows)
for col in data.columns:
    print "Query Results for Column: ", col.name
    for v in col.values:
        print "   ", v
openTable.close()           # we're done
```

- **In future, to get the table back from Original File**

```
orig_table_file = conn.getObject("OriginalFile", attributes={'name': tablename})     # if name is unique
savedTable = conn.c.sf.sharedResources().openTable(orig_table_file._obj)
print "Opened table with row-count:", savedTable.getNumberOfRows()
```

- **Close connection:**

```
# When you are done, close the session to free up server resources.
conn._closeSession()
```

ROIs

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
updateService = conn.getUpdateService()
```

- **Configuration**

```
imageId = 27544
```

- **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.
#
# In this example, we create an ROI with a rectangular shape and attach it to an
# image.
```

```
x = 50
y = 200
width = 100
height = 50
image = conn.getObject("Image", imageId)
theZ = image.getSizeZ() / 2
theT = 0
print "Adding a rectangle at theZ: %s, theT: %s, X: %s, Y: %s, width: %s, height: %s" % \
        (theZ, theT, x, y, width, height)


# create an ROI, link it to Image
roi = omero.model.RoiI()
roi.setImage(image._obj)     # use the omero.model.ImageI that underlies the 'image' wrapper


# create a rectangle shape and add to ROI
rect = omero.model.RectI()
rect.x = rdouble(x)
rect.y = rdouble(y)
rect.width = rdouble(width)
rect.height = rdouble(height)
rect.theZ = rint(theZ)
rect.theT = rint(theT)
rect.textValue = rstring("test-Rectangle")
roi.addShape(rect)


# create an Ellipse shape and add to ROI
ellipse = omero.model.EllipseI()
ellipse.cx = rdouble(y)
ellipse.cy = rdouble(x)
ellipse.rx = rdouble(width)
ellipse.ry = rdouble(height)
ellipse.theZ = rint(theZ)
ellipse.theT = rint(theT)
ellipse.textValue = rstring("test-Ellipse")
roi.addShape(ellipse)


# Save the ROI (saves any linked shapes too)
r = updateService.saveAndReturnObject(roi)
```

- **Retrieve ROIs linked to an Image.**

```
roiService = conn.getRoiService()
result = roiService.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.RectI:
            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['cx'] = s.getCx().getValue()
            shape['cy'] = s.getCy().getValue()
            shape['rx'] = s.getRx().getValue()
            shape['ry'] = s.getRy().getValue()
        elif type(s) == omero.model.PointI:
            shape['type'] = 'Point'
            shape['cx'] = s.getCx().getValue()
            shape['cy'] = s.getCy().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) in (omero.model.MaskI, 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():
            print "  ", key, value,
        print ""
```

- **Remove shape from ROI**

```
result = roiService.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)
```

- **Close connection:**

```
# When you are done, close the session to free up server resources.
conn._closeSession()
```

Delete data

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
```

- **Configuration**

```
projectId = 507         # NB: This will be deleted!
```

- **Load the Project**

```
project = conn.getObject("Project", projectId)
if project is None:
    import sys
    sys.stderr.write("Error: Object does not exist.\n")
    sys.exit(1)


print "\nProject:", project.getName()
```

- **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 = [projectId]
deleteChildren = False
handle = conn.deleteObjects("Project", obj_ids,\
        deleteAnns=True, deleteChildren=deleteChildren)
```

- **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
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
```

- **Close connection:**

```
# When you are done, close the session to free up server resources.
conn._closeSession()
```

Render Images

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
```

- **Configuration**

```
imageId = 27544
```

- **Get thumbnail**

```
# Thumbnail is created using the current rendering settings on the image
image = conn.getObject("Image", imageId)
img_data = image.getThumbnail()
```

```
renderedThumb = Image.open(StringIO(img_data))
#renderedThumb.show()           # shows a pop-up
renderedThumb.save("thumbnail.jpg")
```

- **Get current settings**

```
print "Channel rendering settings:"
for ch in image.getChannels():
    print "Name: ", ch.getLabel()   # if no name, get emission wavelength or index
    print "  Color:", ch.getColor().getHtml()
    print "  Active:", ch.isActive()
    print "  Levels:", ch.getWindowStart(), "-", ch.getWindowEnd()
print "isGreyscaleRenderingModel:", image.isGreyscaleRenderingModel()
```

- **Render each channel as a separate greyscale image**

```
image.setGreyscaleRenderingModel()
sizeC = image.getSizeC()
z = image.getSizeZ() / 2
t = 0
for c in range(1, sizeC + 1):       # Channel index starts at 1
    channels = [c]                  # Turn on a single channel at a time
    image.setActiveChannels(channels)
    renderedImage = image.renderImage(z, t)
    #renderedImage.show()                        # popup (use for debug only)
    renderedImage.save("channel%s.jpg" % c)     # save in the current folder
```

- **Turn 3 channels on, setting their colours**

```
image.setColorRenderingModel()
channels = [1, 2, 3]
colorList = ['F00', None, 'FFFF00']         # do not change colour of 2nd channel
image.setActiveChannels(channels, colors=colorList)
image.setProjection('intmax')               # max intensity projection 'intmean' for mean-intensity
renderedImage = image.renderImage(z, t)     # z and t are ignored for projections
#renderedImage.show()
renderedImage.save("all_channels.jpg")
image.setProjection('normal')               # turn off projection
```

- **Turn 2 channels on, setting levels of the first one**

```
channels = [1, 2]
rangeList = [[100.0, 120.2], [None, None]]
image.setActiveChannels(channels, windows=rangeList)
renderedImage = image.renderImage(z, t, compression=0.5)    # default compression is 0.9
#renderedImage.show()
renderedImage.save("two_channels.jpg")
```

- **Save the current rendering settings**

```
image.saveDefaults()
```

- **Close connection:**

```
# When you are done, close the session to free up server resources.
conn._closeSession()
```

Create Image

- **Create a connection**

```
conn = BlitzGateway(USERNAME, PASSWORD, host=HOST, port=PORT)
conn.connect()
```

- **Configuration**

```
imageId = 27544     # This image must have at least 2 channels
```

- **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
sizeX, sizeY, sizeZ, sizeC, sizeT = 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 planeGen():
    """generator will yield planes"""
    for p in planes:
        yield p


desc = "Image created from a hard-coded arrays"
i = conn.createImageFromNumpySeq(planeGen(), "numpy image",\
        sizeZ, sizeC, sizeT, description=desc, dataset=None)
```

- **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.
zctList = []
image = conn.getObject('Image', imageId)
sizeZ, sizeC, sizeT = image.getSizeZ(), image.getSizeC(), image.getSizeT()
dataset = image.getParent()
pixels = image.getPrimaryPixels()
newSizeC = 1


def planeGen():
    """
    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(sizeZ):              # all Z sections
    for c in range(newSizeC):       # Illustrative purposes only, since we only have 1 channel
        for t in range(sizeT):      # 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 "averag
            newPlane = (channel0 + channel1) / 2    # average of 2 channels
            print "newPlane for z,t:", z, t, newPlane.dtype, newPlane.min(), newPlane.max()
            yield newPlane


desc = "Image created from Image ID: %s by averaging Channel 1 and Channel 2" % imageId
i = conn.createImageFromNumpySeq(planeGen(), "new image",\
        sizeZ, newSizeC, sizeT, description=desc, dataset=dataset)
```

- **Close connection:**

```
# When you are done, close the session to free up server resources.
conn._closeSession()
```

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*.

12.1.2 Blitz Gateway documentation

The epydoc-generated documentation of methods provided by OMERO Gateway is available showing wrapper classes[5].

Specifically, the API for the 'conn' connection wrapper created above is here[6].

When working with OMERO model objects[7] (omero.model.Image etc) the Gateway will wrap these objects in classes such as omero.gateway.ImageWrapper[8] to handle object loading and hierarchy traversal. For example:

```
>>> 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
```

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.

[5] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omero.gateway-module.html
[6] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omero.gateway._BlitzGateway-class.html
[7] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model.html
[8] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omero.gateway._ImageWrapper-class.html

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[9] with the Gateway handling reuse as needed. Stateless services (those retrieved with get… methods E.g. getQueryService[10]) are always reused for each call, E.g. blitzon.getQueryService() whereas stateful services E.g. createRenderingEngine[11] 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[12] 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* framework and it is not expected that stateful services will be maintained on the client for significant time. There is various error-handling functionality 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. We will attempt to document this a little better in due course, but our general advice is to create, use and close the stateful services in the shortest practicable time.

Overwriting and extending omero.gateway classes

When working with omero.gateway[13] or wrapper classes such as omero.gateway.ImageWrapper[14] you might want to add your own functionality or customize an existing one. NB: Note the call to `omero.gateway.refreshWrappers()` to ensure that your subclasses are returned by calls to getObjects() For example:

```
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
```

[9] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/ServiceFactory.html
[10] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/ServiceFactory.html#getQueryService
[11] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/ServiceFactory.html#createRenderingEngine
[12] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omero.gateway-pysrc.html#_BlitzGateway._createProxies
[13] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omero.gateway._BlitzGateway-class.html
[14] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omero.gateway._ImageWrapper-class.html

```
    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).ANNOTATIONLI
            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
    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()
```

This page provides some background information on the OMERO Python client 'gateway' (omero.gateway module) and describes work to improve the API, beginning with the OMERO 4.3 release.

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* framework, to provide connection and data retrieval services to various web clients. However, we have now decided to encourage its use for all access to the OMERO Python API.

Wrapper objects

The Gateway consists of a number of wrapper objects:

Connection wrapper

The BlitzGateway class (see API of development code[15]) 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 as a starting point for retrieving data objects from OMERO.

```
from omero.gateway import *

conn = BlitzGateway("username", "password", host="localhost", port=4064)
conn.connect()
```

[15] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omero.gateway._BlitzGateway-class.html

```
for p in conn.listProjects():
    print p.name
```

Model object wrappers

OMERO model objects, E.g. omero.model.Project, omero.model.Pixels etc (see full list[16]) 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: *about model objects*).

```
import omero
from omero.model import *
from omero.rtypes import rstring
p = omero.model.ProjectI()
p.name = rstring("My Project")    # attributes are all rtypes
print p.getName().getValue()      # getValue() to unwrap the rtype
print p.name.val                  # short-hand
```

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 *
from omero.rtypes import rstring
p = omero.model.ProjectI()
p.setName(rstring("Omero Model Project"))    # attributes are all rtypes
print p.getName().getValue()                 # getValue() to unwrap the rtype
print p.name.val                             # short-hand

from omero.gateway import *
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).

E.g.

```
# connect as above
for p in conn.listProjects():
    print p.name
    for dataset in p.listChildren():    # lazy loading of datasets, wrapped in DatasetWrapper
        print "Dataset", d.name
```

Wrapper coverage

The OMERO data model has a large number of objects, not all of which are used by the *OMERO.web framework* framework. For this reason, the Blitz gateway (which was originally built for *OMERO.web 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[17]. As more functionality is provided by the Blitz Gateway, the coverage of object wrappers will increase accordingly.

[16] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model.html
[17] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omero.gateway-module.html

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

12.2 OMERO Command Line Interface

See also:

OMERO Command Line Interface User documentation on the Command Line Interface

OMERO Command Line Interface System Administrator documentation for the Command Line Interface

12.2.1 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. Alternatively, the "–path" argument can be used to point to other plugin files or directories.

Thread-safety

The `omero.cli.CLI` should be considered thread-*un*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 Other extensions to OMERO

The following should be considered a design document. A portion of the functionality is included in the milestone OMERO-Beta4[18] and later releases, but more functionality will continually be added. If you would like to request a particular function, please open a ticket.

12.2.2 Design plans

General notes:

- `bin/omero` will find its installation. Therefore, to install OMERO it is only necessary to unpack the bundle, and put `bin/omero` somewhere on your path.
- 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`.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

12.3 OMERO Java language bindings

Using the Ice Java language mapping[19] from ZeroC[20], OMERO provides access to your data within an *OMERO.blitz* server from Java code.

[18] http://trac.openmicroscopy.org.uk/ome/milestone/OMERO-Beta4
[19] http://zeroc.com/doc/Ice-3.3.0/manual/Hello.4.4.html
[20] http://www.zeroc.com

12.3.1 Using the `omero_client.jar`

The `omero_client.jar` is a combination of all necessary Java OMERO class as well as the Ice classes needed to write a complete Java client for *OMERO.blitz*.

The library is placed under OMERO_HOME/dist/lib/client by the build, or is alternatively available from Jenkins here[21]. To use *OMERO Java language bindings*, setup you will need to setup your CLASSPATH:

```
CLASSPATH=path/omero_client.jar
javac mycode
```

12.3.2 Extended classpath

To access all the functionality available in omero_client.jar or to use the importer, you will need more jar files. To see all the current requirements, take a look at the builds on jenkins[22], or alternatively examine the dependencies in the ivy.xml files (e.g. components/tools/OmeroImporter/ivy.xml[23])

12.3.3 Connect to OMERO

- **Connect to the server**. Remember to close the session.

```
client client = new client(hostName, port);
ServiceFactoryPrx entry = client.createSession(userName, password);
// if you want to have the data transfer encrypted then you can
// use the entry variable otherwise use the following
client unsecureClient = client.createClient(false);
ServiceFactoryPrx entryUnencrypted = unsecureClient.getSession();

//Retrieve the user id.
long userId = entryUnencrypted.getAdminService().getEventContext().userId;

long groupId = entryUnencrypted.getAdminService().getEventContext().groupId;
```

- **Close connection**. **IMPORTANT**

```
client.closeSession();
//if unsecure client exists.
if (unsecureClient != null) unsecureClient.closeSession();
```

12.3.4 Read data

The IContainer service provides method to load the data management hierarchy in OMERO. 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.

```
IContainerPrx proxy = entryUnencrypted.getContainerService();
ParametersI param = new ParametersI();
long userId = entryUnencrypted.getAdminService().getEventContext().userId;
param.exp(omero.rtypes.rlong(userId));
param.leaves(); //indicate to load the images
```

[21] http://ci.openmicroscopy.org/job/OMERO-trunk/lastSuccessfulBuild/
[22] http://ci.openmicroscopy.org/
[23] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroImporter/ivy.xml

```
//param.noLeaves(); //no images loaded, this is the default value.
List<IObject> results = proxy.loadContainerHierarchy(
Project.class.getName(), new ArrayList<Long>(), param);
//You can directly interact with the IObject or the Pojos object.
//Follow interaction with the Pojos.
Iterator<IObject> i = results.iterator();
ProjectData project;
Set<DatasetData> datasets;
Iterator<DatasetData> j;
DatasetData dataset;
while (i.hasNext()) {
  project = new ProjectData((Project) i.next());
  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.

```
IContainerPrx proxy = entryUnencrypted.getContainerService();
ParametersI param = new ParametersI();
long userId = entryUnencrypted.getAdminService().getEventContext().userId;
param.exp(omero.rtypes.rlong(userId));

//indicate to load the images
param.leaves();
List<IObject> results = proxy.loadContainerHierarchy(Dataset.class.getName(), new ArrayList<Long>(), pa
//You can directly interact with the IObject or the Pojos object.
//Follow interaction with the Pojos.
Iterator<IObject> i = results.iterator();
DatasetData dataset;
Set<ImageData> images;
Iterator<ImageData> j;
ImageData image;
while (i.hasNext()) {
  dataset = new DatasetData((Dataset) i.next());
  images = dataset.getImages();
  j = images.iterator();
  while (j.hasNext()) {
    image = j.next();
    //Do something
  }
}
```

- **Retrieve the Images** contained in a Dataset.

```
IContainerPrx proxy = entryUnencrypted.getContainerService();
ParametersI param = new ParametersI();
param.leaves(); //indicate to load the images

List<IObject> results = proxy.loadContainerHierarchy(Dataset.class.getName(), Arrays.asList(datasetId),

if (results.size() == 0) return;
//You can directly interact with the IObject or the Pojos object.
//Follow interaction with the Pojos.
```

```
DatasetData dataset = new DatasetData((Dataset) results.get(0));
Set<ImageData> images = dataset.getImages();
Iterator<ImageData> j = images.iterator();
ImageData image;
while (j.hasNext()) {
  image = j.next();
  //Do something
}
```

- **Retrieve an Image** if the identifier is known.

```
IContainerPrx proxy = entryUnencrypted.getContainerService();
List<Image> results = proxy.getImages(Image.class.getName(), Arrays.asList(imageId), new ParametersI());

if (results.size() == 0) return;
//You can directly interact with the IObject or the Pojos object.
//Follow interaction with the Pojos.
ImageData image = new ImageData(results.get(0));
```

- **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.

```
IContainerPrx proxy = entryUnencrypted.getContainerService();
ParametersI param = new ParametersI();
long userId = entryUnencrypted.getAdminService().getEventContext().userId;
param.exp(omero.rtypes.rlong(userId));

List<IObject> results = proxy.loadContainerHierarchy(Screen.class.getName(), new ArrayList(), param);
//You can directly interact with the IObject or the Pojos object.
//Follow interaction with the Pojos.
Iterator<IObject> i = results.iterator();
ScreenData screen;
Set<PlateData> plates;
Iterator<PlateData> j;
PlateData plate;
while (i.hasNext()) {
  screen = new ScreenData((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. You will have to use the `findAllByQuery` method from the `IQuery` service.

```
IQueryPrx proxy = entryUnencrypted.getQueryService();
StringBuilder sb = new StringBuilder();
ParametersI param = new ParametersI();
param.addLong("plateID", plateId);
sb.append("select well from Well as well ");
sb.append("left outer join fetch well.plate as pt ");
sb.append("left outer join fetch well.wellSamples as ws ");
sb.append("left outer join fetch ws.plateAcquisition as pa ");
sb.append("left outer join fetch ws.image as img ");
sb.append("left outer join fetch img.pixels as pix ");
sb.append("left outer join fetch pix.pixelsType as pt ");
sb.append("where well.plate.id = :plateID");
if (plateAcquisitionId > 0) {
  sb.append(" and pa.id = :acquisitionID");
  param.addLong("acquisitionID", plateAcquisitionId);
}
List<IObject> results = proxy.findAllByQuery(sb.toString(), param);
Iterator<IObject> i = results.iterator();
WellData well;
while (i.hasNext()) {
  well = new WellData((Well) i.next());
  //Do something
}
```

12.3.5 Raw data access

- **Retrieve a given plane**.

This is useful when you need the pixels intensity.

```
//To retrieve the image, see above.
PixelsData pixels = image.getDefaultPixels();
int sizeZ = pixels.getSizeZ();
int sizeT = pixels.getSizeT();
int sizeC = pixels.getSizeC();
long pixelsId = pixels.getId();
RawPixelsStorePrx store = entryUnencrypted.createRawPixelsStore();
store.setPixelsId(pixelsId, false);
for (int z = 0; z < sizeZ; z++) {
  for (int t = 0; t < sizeT; t++) {
    for (int c = 0; c < sizeC; c++) {
      byte[] plane = store.getPlane(z, c, t);
      //Do something
    }
  }
}
store.close();
```

- **Retrieve a given tile.**

```
//To retrieve the image, see above.
PixelsData pixels = image.getDefaultPixels();
int sizeZ = pixels.getSizeZ();
int sizeT = pixels.getSizeT();
int sizeC = pixels.getSizeC();
long pixelsId = pixels.getId();
RawPixelsStorePrx store = entryUnencrypted.createRawPixelsStore();
store.setPixelsId(pixelsId, false);
//tile is the top-left corner
int x = 0;
```

```
int y = 0;
int width = pixels.getSizeX()/2;
int height = pixels.getSizeY()/2;
for (int z = 0; z < sizeZ; z++) {
  for (int t = 0; t < sizeT; t++) {
    for (int c = 0; c < sizeC; c++) {
      byte[] plane = store.getTile(z, c, t, x, y, width, height);
      //Do something
    }
  }
}
store.close();
```

- **Retrieve a given stack.**

This is useful when you need the pixels intensity.

```
//To retrieve the image, see above.
PixelsData pixels = image.getDefaultPixels();
int sizeT = pixels.getSizeT();
int sizeC = pixels.getSizeC();
long pixelsId = pixels.getId();
RawPixelsStorePrx store = entryUnencrypted.createRawPixelsStore();
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
  }
}
store.close();
```

- **Retrieve a given hypercube.**

This is useful when you need the pixels intensity.

```
//To retrieve the image, see above.
PixelsData pixels = image.getDefaultPixels();
long pixelsId = pixels.getId();
RawPixelsStorePrx store = entryUnencrypted.createRawPixelsStore();
store.setPixelsId(pixelsId, false);
// offset values in each dimension XYZCT
List<Integer> offset = new ArrayList<Integer>();
offset.add(0);
offset.add(0);
offset.add(0);
offset.add(0);
offset.add(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>();
step.add(1);
step.add(1);
```

```
step.add(1);
step.add(1);
step.add(1);
byte[] values = store.getHypercube(offset, size, step);
//Do something
store.close();
```

12.3.6 Write data

- **Create a dataset and link it to an existing project.**

```
//Using IObject directly
Dataset dataset = new DatasetI();
dataset.setName(omero.rtypes.rstring("new Name 1"));
dataset.setDescription(omero.rtypes.rstring("new description 1"));

//Using pojo object
DatasetData datasetData = new DatasetData();
datasetData.setName("new Name 2");
datasetData.setDescription("new description 2");

ProjectDatasetLink link = new ProjectDatasetLinkI();
link.setChild(dataset);
link.setParent(new ProjectI(projectId, false));
IObject r = entryUnencrypted.getUpdateService().saveAndReturnObject(link);

//With pojo
link = new ProjectDatasetLinkI();
link.setChild(datasetData.asDataset());
link.setParent(new ProjectI(projectId, false));
r = entryUnencrypted.getUpdateService().saveAndReturnObject(link);
```

- **Create a tag (tag annotation) and link it to an existing project.**

```
//Using the IObject.
TagAnnotation tag = new TagAnnotationI();
tag.setTextValue(omero.rtypes.rstring("new tag 1"));
tag.setDescription(omero.rtypes.rstring("new tag 1"));

//Using the Pojo
TagAnnotationData tagData = new TagAnnotationData("new tag 2");
tagData.setTagDescription("new tag 2");

//link project and annotation
ProjectAnnotationLink link = new ProjectAnnotationLinkI();
link.setChild(tag);
link.setParent(new ProjectI(projectId, false));

IObject r = entryUnencrypted.getUpdateService().saveAndReturnObject(link);

//With pojo

link = new ProjectAnnotationLinkI();
link.setChild(tagData.asAnnotation());
link.setParent(new ProjectI(projectId, false));
r = entryUnencrypted.getUpdateService().saveAndReturnObject(link);
```

- **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`.

```
// To retrieve the image see above.
int INC = 262144;
File file = new File(fileToUpload);
String name = file.getName();
String absolutePath = file.getAbsolutePath();
String path = absolutePath.substring(0,
  absolutePath.length()-name.length());

IUpdatePrx iUpdate = entryUnencrypted.getUpdateService(); // service used to write object
// 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()));
originalFile.setSha1(omero.rtypes.rstring(generatedSha1));
originalFile.setMimetype(omero.rtypes.rstring(fileMimeType)); // or "application/octet-stream"
// now we save the originalFile object
originalFile = (OriginalFile) iUpdate.saveAndReturnObject(originalFile);

// Initialize the service to load the raw data
RawFileStorePrx rawFileStore = entryUnencrypted.createRawFileStore();
rawFileStore.setFileId(originalFile.getId().getValue());

FileInputStream stream = new FileInputStream(file);
long pos = 0;
int rlen;
byte[] buf = new byte[INC];
ByteBuffer bbuf;
while ((rlen = stream.read(buf)) > 0) {
  rawFileStore.write(buf, pos, rlen);
  pos += rlen;
  bbuf = ByteBuffer.wrap(buf);
  bbuf.limit(rlen);
}
stream.close();

originalFile = rawFileStore.save();
// Important to close the service
rawFileStore.close();

//now we have an original File in the database and raw data uploaded.
// We now need to link the Original file to the image using
// the File annotation object. This is the way to do it.
FileAnnotation fa = new FileAnnotationI();
fa.setFile(originalFile);
fa.setDescription(omero.rtypes.rstring(description));
fa.setNs(omero.rtypes.rstring(NAME_SPACE_TO_SET)); // The name space you have set to identify the file a

// save the file annotation.
fa = (FileAnnotation) iUpdate.saveAndReturnObject(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) iUpdate.saveAndReturnObject(link);
// To attach to a Dataset use DatasetAnnotationLink;
```

- **Load all the annotations with a given namespace linked to images.**

```
long userId = entryUnencrypted.getAdminService().getEventContext().userId;
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 = entryUnencrypted.getMetadataService();
// retrieve the annotations linked to images, for datasets use: omero.model.Dataset.class
List<Annotation> annotations = proxy.loadSpecifiedAnnotations(FileAnnotation.class.getName(), nsToInclud
//Do something with annotations.
```

- **Read the attachment**.

First load the annotations, cf. above.

```
Iterator<Annotation> j = annotations.iterator();
Annotation annotation;
FileAnnotationData fa;
RawFileStorePrx store = entryUnencrypted.createRawFileStore();
int index = 0;
File file = new File(downloadFileName); //This file should be there.
FileOutputStream stream = new FileOutputStream(file);
OriginalFile of;
while (j.hasNext()) {
  annotation = j.next();
  if (annotation instanceof FileAnnotation && index == 0) { //read the first one.
    fa = new FileAnnotationData((FileAnnotation) annotation);
    //The id of the original file
    of = getOriginalFile(fa.getFileID());
    store.setFileId(fa.getFileID());
    int offset = 0;
    long size = of.getSize().getValue();
    //name of the file
    //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)));
    stream.close();
    }
    break;
  }
}

store.close();
```

12.3.7 How to use OMERO tables

- **Create a table**.

In the following example, we create a table with 2 columns.

```
/**
 * Creates a number of empty rows.
 *
 * @param rows The number of rows.
 * @return See above.
```

```
 */
private Column[] createColumns(int rows)
{
  Column[] newColumns = new Column[2];
  newColumns[0] = new LongColumn("Uid", "", new long[rows]);
  newColumns[1] = new LongColumn("MyLongColumn", "",
                new long[rows]);
  return newColumns;
}

int rows = 1;
String name = UUID.randomUUID().toString();
Column[] columns = createColumns(rows);

//create a new table.
TablePrx table = entryUnencrypted.sharedResources().newTable(1, name);

//initialize the table
table.initialize(columns);
//add data to the table.
rows = 2;
Column[] newRow = createColumns(rows);

LongColumn uids = (LongColumn) newRow[0];
LongColumn myLongs = (LongColumn) newRow[1];
for (int i = 0; i < rows; i++) {
  uids.values[i] = i;
  myLongs.values[i] = i;
}

table.addData(newRow);
OriginalFile file = table.getOriginalFile(); // if you need to interact with the table
```

- **Read the contents of the table.**

```
file = new OriginalFileI(file.getId(), false);
table = entryUnencrypted.sharedResources().openTable(file);

//read headers
Column[] cols = table.getHeaders();

for (int i = 0; i < cols.length; i++) {
  String colName = cols[i].name;
}

// Depending on size of table, you may only want to read some blocks.
long[] columnsToRead = new long[cols.length];
for (int i = 0; i < cols.length; i++) {
  columnsToRead[i] = i;
}

// The number of columns we wish to read.
long[] rowSubset = new long[(int) (table.getNumberOfRows()-1)];
for (int j = 0; j < rowSubset.length; j++) {
  rowSubset[j] = j;
}
Data data = table.slice(columnsToRead, rowSubset); // read the data.
cols = data.columns;
for (int j = 0; j < cols.length; j++) {
  Column c = cols[j];
}
table.close();
```

12.3.8 ROIs

To learn about the model see developers/roi.html[24]. Note that annotation can be linked to ROI.

- **Create ROI.**

In this example, we create an ROI with a rectangular shape and attach it to an image.

```
//to retrieve he image see above.
Roi roi = new RoiI();
roi.setImage(image);
Rect rect;
rect = new RectI();
rect.setX(omero.rtypes.rdouble(10));
rect.setY(omero.rtypes.rdouble(10));
rect.setWidth(omero.rtypes.rdouble(10));
rect.setHeight(omero.rtypes.rdouble(10));
rect.setTheZ(omero.rtypes.rint(0));
rect.setTheT(omero.rtypes.rint(0));

//Add the shape
roi.addShape(rect);

//Create an ellipse.
EllipseI ellipse = new EllipseI();
ellipse.setCx(omero.rtypes.rdouble(10));
ellipse.setCy(omero.rtypes.rdouble(10));
ellipse.setRx(omero.rtypes.rdouble(10));
ellipse.setRy(omero.rtypes.rdouble(10));
ellipse.setTheZ(omero.rtypes.rint(0));
ellipse.setTheT(omero.rtypes.rint(0));
ellipse.setTextValue(omero.rtypes.rstring("ellipse text"));

//Add the shape
roi.addShape(ellipse);
//Save ROI and shape
roi = (Roi) entryUnencrypted.getUpdateService().saveAndReturnObject(roi);

//now check that the shape has been added.
ROIData roiData = new ROIData(roi);
//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;
   //Handle rectangle
  } else if (shape instanceof EllipseData) {
    EllipseData ellipseData = (EllipseData) shape;
    //Handle ellipse
  } else if (shape instanceof LineData) {
    LineData lineData = (LineData) shape;
    //Handle line
  } else if (shape instanceof PointData) {
    PointData pointData = (PointData) shape;
    //Handle point
  }
}
```

[24] http://www.openmicroscopy.org/site/support/ome-model/developers/roi.html

- **Retrieve ROIs linked to an Image.**

```
// Retrieve the roi linked to an image
RoiResult r = entryUnencrypted.getRoiService().findByImage(image.getId().getValue(), new RoiOptions());
if (r == null) return;
List<Roi> rois = r.rois;
List<Shape> list;
Iterator<Roi> j = rois.iterator();
while (j.hasNext()) {
  roi = j.next();
  list = roi.copyShapes();
  //Do something
}
```

- **Remove a shape from ROI.**

```
// Retrieve the roi linked to an image
RoiResult r = entryUnencrypted.getRoiService().findByImage(image.getId().getValue(), new RoiOptions());
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.
  if (list.size() > 0) {
    roi.removeShape(list.get(0));
    //update the roi.
    entryUnencrypted.getUpdateService().saveAndReturnObject(roi);
  }
}
```

12.3.9 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.

```
//First create an image.
Image img = new ImageI();
img.setName(omero.rtypes.rstring("image1"));
img.setDescription(omero.rtypes.rstring("descriptionImage1"));
img.setAcquisitionDate(omero.rtypes.rtime(1000000));
img = (Image) entryUnencrypted.getUpdateService().saveAndReturnObject(img);

DeleteCommand[] cmds = new DeleteCommand[1];
//Command to delete the image.
cmds[0] = new DeleteCommand("/Image", img.getId().getValue(), null);
DeleteHandlePrx handle = entryUnencrypted.getDeleteService().queueDelete(cmds);

//If you want to interact with call-back and handle.
DeleteCallbackI cb = new DeleteCallbackI(client, handle);
DeleteReport[] reports = handle.report();
for (int i = 0; i < reports.length; i++) {
  DeleteReport report = reports[i];
  String error = report.error;
}
```

12.3.10 Render Images

- **Initialize the rendering engine and render an image.**

```
//See above how to load the image.
PixelsData pixels = image.getDefaultPixels();
long pixelsId = pixels.getId();
RenderingEnginePrx proxy = entryUnencrypted.createRenderingEngine();
proxy.lookupPixels(pixelsId);
if (!(proxy.lookupRenderingDef(pixelsId))) {
  proxy.resetDefaults();
  proxy.lookupRenderingDef(pixelsId);
}
proxy.load();
// Now can interact with the rendering engine.
proxy.setActive(0, Boolean.valueOf(false));
// to render the image uncompressed
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
ByteArrayInputStream stream = new ByteArrayInputStream(compressed);
BufferedImage image = ImageIO.read(stream);

// Close
proxy.close();
```

- **Retrieve thumbnails.**

```
//See above how to load the image.
PixelsData pixels = image.getDefaultPixels();
ThumbnailStorePrx store = entryUnencrypted.createThumbnailStore();
PixelsData pixels = image.getDefaultPixels();
Map<Long, byte[]> map = store.getThumbnailByLongestSideSet(
   omero.rtypes.rint(96), Arrays.asList(pixels.getId()));
//Convert the byte array
Entry entry;
Iterator i = map.entrySet().iterator();
ByteArrayInputStream stream;
//Create a buffered image to display
Map<Long, BufferedImage> results = new HashMap<Long, BufferedImage>();
while (i.hasNext()) {
  entry = (Entry) i.next();
  stream = new ByteArrayInputStream((byte[]) entry.getValue());
  results.put((Long) entry.getKey(), ImageIO.read(stream));
}
```

12.3.11 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 = entryUnencrypted.createRawPixelsStore();
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));
    }
}

//Better to close to free space.
store.close();

//Now we are going to create the new image.
IPixelsPrx proxy = entryUnencrypted.getPixelsService();

//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,
        "From Image ID: "+image.getId());
if (idNew == null)
    throw new Exception("New image could not be created.");
ImageData newImage = loadImage(idNew.getValue());

//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));
entryUnencrypted.getUpdateService().saveAndReturnObject(link);

//Write the data.
store = entryUnencrypted.createRawPixelsStore();
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();

store.close();
```

12.3.12 Further information

For the details behind writing, configuring, and executing a client, please see *Working with OMERO*.

See also:

ZeroC[25], *OMERO.grid*, *OmeroTools*, *OMERO Application Programming Interface*

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

12.4 OMERO Matlab language bindings

See *Developing OMERO clients* and *OME-Remote Objects*, for an introduction to **Object**.

12.4.1 Installing the OMERO.matlab toolbox

- Download the latest released version from the download page[26] . For the latest build, go here[27], and download the OMERO.matlab zip file.
- 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…)
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, port);
session = client.createSession(user, password);
client.enableKeepAlive(60);
images = getImages(session, ids);
client.closeSession();
```

[25] http://www.zeroc.com
[26] http://downloads.openmicroscopy.org/latest/omero4/
[27] http://ci.openmicroscopy.org/job/OMERO-trunk/lastSuccessfulBuild/

12.4.2 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 with an output 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 same parameters to `loadOmero;` that you would pass to `omero.client`:

```
>> omero_client_1 = loadOmero('localhost');
>> omero_client_2 = omero.client('localhost');
```

Or, if you want a session created directly, the following are equivalent:

```
>> [client1, session1] = loadOmero('localhost');
>> client2 = loadOmero('localhost');
>> session2 = client2.createSession()
```

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[28] directly or modify it to your liking.

```
>> [c,s] = loadOmero;
>> t = omeroKeepAlive(c); % Create a 60-second timer and starts it
>> …
>> delete(t);             % Disable the keep-alive
```

Alternatively, you can use the Java-based `enableKeepAlive` method, but it is not configurable from within Matlab:

```
c.enableKeepAlive(60); % Call session.keepAlive() every 60 seconds
c.closeSession();      % Close session to end the keep-alive
```

Creating an unencrypted session

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:

[28] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/omeroKeepAlive.m

```
client.closeSession();
clear client1;
clear session1;
```

If you created an unencrypted session, you will need to close the unsecure session as well:

```
client.closeSession();
unsecureClient.closeSession();
```

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:

```
>> [c,s] = loadOmero(arg1,arg2);
>> c = omero.client(arg3,arg4);
>> s = c.createSession();
```

12.4.3 Reading data

The `IContainer` service provides methods to load the data management hierarchy in OMERO – projects, datasets… A list of examples follows indicating how to load projects, datasets, screens…

- **Projects**

The projects owned by the user currently logged in can be retrieved using the getProjects[29] function:

```
projects = getProjects(session)
```

If the project identifiers are known, they can be specified as:

```
projects = getProjects(session, ids)
```

If the projects contain datasets, the datasets will automatically be loaded:

```
for j = 1 : numel(projects)
    datasetsList = projects(j).linkedDatasetList;
    for i = 0:datasetsList.size()-1,
        d = datasetsList.get(i);
    end
end
```

If the datasets contain images, the images will automatically be loaded:

```
imageList = projects(1).linkedDatasetList.get(0).linkedImageList;
```

To avoid loading the whole graph (projects, datasets, images), pass *false* as a second optional argument. Only datasets will be loaded:

```
unloadedProjects = getProjects(session, ids, false)
```

- **Datasets**

The datasets owned by the user currently logged in can be retrieved using the getDatasets[30] function:

```
datasets = getDatasets(session)
```

If the dataset identifiers are known, they can be specified as:

```
datasets = getDatasets(session, ids)
```

If the datasets contain images, the images will automatically be loaded:

```
imageList = datasets(1).linkedImageList;
```

To avoid loading the images, pass *false* as a second optional argument:

```
unloadedDatasets = getDatasets(session, ids, false)
```

- **Images**

[29] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/io/getProjects.m
[30] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/io/getDatasets.m

All the images owned by the user currently logged in can be retrieved using the getImages[31] function:

```
images = getImages(session)
```

If the image identifiers are known, they can be specified as:

```
images = getImages(session, ids)
```

All the images contained in a subset of datasets of known identifiers `datasetsIds` can be returned 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 using:

```
projectImages = getImages(session, 'project', projectIds)
```

The `Image-Pixels` model 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 user currently logged in can be retrieved using the getScreens[32] function:

```
screens = getScreens(session)
```

If the screen identifiers are known, they can be specified as:

```
screens = getScreens(session, ids)
```

Note that the wells are not loaded. The plate objects can be accessed using:

```
for j = 1 : numel(screens),
platesList = screens(j).linkedPlateList;
for i = 0:platesList.size()-1,
    plate = platesList.get(i);
    plateAcquisitionList = plate.copyPlateAcquisitions();
    for k = 0:plateAcquisitionList.size()-1,
        pa = plateAcquisitionList.get(i);
    end
end
```

- **Plates**

The plates owned by the user currently logged in can be retrieved using the getPlates[33] function:

[31] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/io/getImages.m
[32] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/io/getScreens.m
[33] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/io/getPlates.m

```
plates = getPlates(session)
```

If the plate identifiers are known, they can be specified as:

```
plates = getPlates(session, ids)
```

- **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)], []);
for j = 0:wellList.size()-1,
    well = wellList.get(j);
    wellsSampleList = well.copyWellSamples();
    well.getId().getValue()
    for i = 0:wellsSampleList.size()-1,
        ws = wellsSampleList.get(i);
        ws.getId().getValue()
        pa = ws.getPlateAcquisition();
    end
end
```

12.4.4 Raw data access

You can retrieve data, plane by plane or retrieve a stack.

- **Plane**

The plane of an input image at coordinates (z, c, t) can be retrieved using the getPlane[34] 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) and of dimensions (w, h) can be retrieved using the getTile[35] function:

```
tile = getTile(session, image, z, c, t, x, y, w, h);
```

Alternatively, the image identifier can be passed to the function:

[34] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/image/getPlane.m
[35] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/image/getTile.m

```
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[36] function:

```
stack = getStack(session, image, c, t);
```

Alternatively, the image identifier can be passed to the function:

```
stack = getStack(session, imageID, c, t);
```

- **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;
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…
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();
```

12.4.5 Annotations

The following table lists all OMERO.matlab functions used to manipulate annotations from OMERO:

[36] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/image/getStack.m

	Tag	File	Comment	XML
Get by identifier	getTagAnnotations[65]	getFileAnnotations[66]	getCommentAnnotations[67]	getXmlAnnotations[68]
Linked to images	getImageTagAnnotations[69]	getImageFileAnnotations[70]	getImageCommentAnnotations[71]	getImageXmlAnnotations[72]
Linked to datasets	getDatasetTagAnnotations[73]	getDatasetFileAnnotations[74]	getDatasetCommentAnnotations[75]	getDatasetXmlAnnotations[76]
Linked to projects	getProjectTagAnnotations[77]	getProjectFileAnnotations[78]	getProjectCommentAnnotations[79]	getProjectXmlAnnotations[80]
Linked to screens	getScreenTagAnnotations[81]	getScreenFileAnnotations[82]	getScreenCommentAnnotations[83]	getScreenXmlAnnotations[84]
Linked to plates	getPlateTagAnnotations[85]	getPlateFileAnnotations[86]	getPlateCommentAnnotations[87]	getPlateXmlAnnotations[88]
Write	writeTagAnnotation[89]	writeFileAnnotation[90]	writeCommentAnnotation[91]	writeXmlAnnotation[92]

[37] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getTagAnnotations.m
[38] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getFileAnnotations.m
[39] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getCommentAnnotations.m
[40] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getXmlAnnotations.m
[41] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getImageTagAnnotations.m
[42] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getImageFileAnnotations.m
[43] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getImageCommentAnnotations.m
[44] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getImageXmlAnnotations.m
[45] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getDatasetTagAnnotations.m
[46] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getDatasetFileAnnotations.m
[47] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getDatasetCommentAnnotations.m
[48] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getDatasetXmlAnnotations.m
[49] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getProjectTagAnnotations.m
[50] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getProjectFileAnnotations.m
[51] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getProjectCommentAnnotations.m
[52] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getProjectXmlAnnotations.m
[53] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getScreenTagAnnotations.m
[54] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getScreenFileAnnotations.m
[55] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getScreenCommentAnnotations.m
[56] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getScreenXmlAnnotations.m
[57] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getPlateTagAnnotations.m
[58] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getPlateFileAnnotations.m
[59] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getPlateCommentAnnotations.m
[60] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getPlateXmlAnnotations.m
[61] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/writeTagAnnotation.m
[62] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/writeFileAnnotation.m
[63] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/writeCommentAnnotation.m
[64] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/writeXmlAnnotation.m
[65] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getTagAnnotations.m
[66] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getFileAnnotations.m
[67] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getCommentAnnotations.m
[68] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getXmlAnnotations.m
[69] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getImageTagAnnotations.m
[70] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getImageFileAnnotations.m
[71] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getImageCommentAnnotations.m
[72] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getImageXmlAnnotations.m
[73] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getDatasetTagAnnotations.m
[74] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getDatasetFileAnnotations.m
[75] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getDatasetCommentAnnotations.m
[76] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getDatasetXmlAnnotations.m
[77] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getProjectTagAnnotations.m
[78] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getProjectFileAnnotations.m
[79] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getProjectCommentAnnotations.m
[80] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getProjectXmlAnnotations.m
[81] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getScreenTagAnnotations.m
[82] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getScreenFileAnnotations.m
[83] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getScreenCommentAnnotations.m
[84] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getScreenXmlAnnotations.m
[85] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getPlateTagAnnotations.m
[86] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getPlateFileAnnotations.m
[87] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getPlateCommentAnnotations.m
[88] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getPlateXmlAnnotations.m
[89] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/writeTagAnnotation.m
[90] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/writeFileAnnotation.m
[91] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/writeCommentAnnotation.m

- **Reading annotations**

If the identifier of the annotation of a given type is known, the annotation can be retrieved from the server using the corresponding function, e.g. for tags using the getTagAnnotations[93] function:

```
tagAnnotations = getTagAnnotations(session, tagIds);
```

Alternatively, the annotations of a given type linked to a given object can be retrieved using the corresponding function, e.g. to retrieve all tags linked to images getImageTagAnnotations[94] function:

```
tagAnnotations = getImageTagAnnotations(session, imageIds);
```

- **Reading file annotations**

The content of a file annotation can be downloaded to local disk using the getFileAnnotationContent[95] 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 annotations**

New annotations can be created using the corresponding `write*Annotation` function (see table above). Existing annotations can be linked to existing objects on the server using the linkAnnotation[96] function.

For example, to create a new tag annotation `tag_name` and attach it to the image `image_id`:

```
tagAnnotation = writeTagAnnotation(session, tag_name);
link = linkAnnotation(session, tagAnnotation, 'Image', image_id);
```

To create a file annotations from the content of a `local_file_path` and attach it to the image `image_id`:

```
fileAnnotation = writeFileAnnotation(session, local_file_path);
link = linkAnnotation(session, fileAnnotation, 'Image', image_id);
```

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[97] 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);
```

12.4.6 Writing data

- **Create a Dataset** and link it to an existing project.

[92] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/writeXmlAnnotation.m
[93] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getTagAnnotations.m
[94] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getImageTagAnnotations.m
[95] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/getFileAnnotationContent.m
[96] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/linkAnnotation.m
[97] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/annotations/updateFileAnnotation.m

```
dataset = omero.model.DatasetI;
dataset.setName(omero.rtypes.rstring(char('name dataset')));
dataset.setDescription(omero.rtypes.rstring(char('description dataset')));

% Link Dataset and Project

link = omero.model.ProjectDatasetLinkI;
link.setChild(dataset);
link.setParent(omero.model.ProjectI(projectId, false));

session.getUpdateService().saveAndReturnObject(link);
```

12.4.7 How to use OMERO tables

- **Create a table**. In the following example, a table is created with 2 columns.

```
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);
% Add data to the table.
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);
table.addData(data);
file = table.getOriginalFile(); % if you need to interact with the table
```

- **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.
```

12.4.8 ROIs

To learn about the model see developers/roi.html[98]. Note that annotation can be linked to ROI.

- **Creating ROI**

This example creates a ROI with two shapes, a rectangle and an ellipse, and attaches it to an image:

```
% First create a rectangular shape.
rectangle = createRectangle(0, 0, 10, 20);
% Indicate on which plane to attach the shape
setShapeCoordinates(rectangle, 0, 0, 0);

% First create an ellipse shape.
ellipse = createEllipse(0, 0, 10, 20);
% Indicate on which plane to attach the shape
setShapeCoordinates(ellipse, 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);

% 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[99] 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.Rect'))
           rectangle = shape;
           rectangle.getX().getValue()
        elseif (isa(shape, 'omero.model.Ellipse'))
           ellipse = shape;
           ellipse.getCx().getValue()
        elseif (isa(shape, 'omero.model.Point'))
           point = shape;
           point.getX().getValue();
        elseif (isa(shape, 'omero.model.Line'))
           line = shape;
           line.getX1().getValue();
```

[98] http://www.openmicroscopy.org/site/support/ome-model/developers/roi.html

```
        end
    end
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
```

12.4.9 Deleting data

It is possible to delete projects, datasets, images, ROIs… 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[100] function:

```
deleteImages(session, imageIds);
```

See also:

deleteProjects[101], **deleteDatasets**[102], **deleteScreens**[103], **deletePlates**[104] Utility functions to delete objects

12.4.10 Rendering images

The RenderImages.m[105] example script shows how to initialize the rendering engine and render an image.

12.4.11 Creating Image

The CreateImage.m[106] 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

12.5 OMERO C++ language bindings

Using the Ice C++ language mapping[107] from ZeroC[108], OMERO provides native access to your data from C++ code. The `build-cpp` build target produces a platform-dependent shared library which can be linked to your application.

Binaries are not provided, therefore it will be necessary for you to compile your own.

[100] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroM/src/delete/deleteImages.m
[105] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/Training/matlab/RenderImages.m
[106] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/Training/matlab/CreateImage.m
[107] http://doc.zeroc.com/display/Ice/Hello+World+Application
[108] http://www.zeroc.com

12.5.1 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

12.5.2 Restrictions

Ice 3.3 and Ice 3.4 will only build with GCC versions *older* than 4.6 (they contain broken headers which newer GCC versions will not parse). GCC 4.4 is tested and recommended.

Ice 3.5 will build with any GCC version up to 4.8, the latest stable version; later versions may work, but are untested.

The version of GCC and/or Ice provided on your system should be compatible, but if you are restricted to a particular version of GCC or Ice, you may need to obtain or build a compatible version of Ice or GCC, respectively.

12.5.3 Preparing to build

Begin by following the instructions under *Checking out the source code* on acquiring the source code. Be **sure** that the git branch you are using matches the version of your server!

Set the `ICE_HOME` environment variable for your installation. This location varies depending upon the installation location and Ice version in use. Some possible locations for the 3.5.0 version of Ice follow. Note these are just examples; you need to adjust them for the Ice installation path and version in use.

- Ice built from source and installed into `/opt`:

  ```
  export ICE_HOME=/opt/Ice-3.5.0
  ```

- Ice installed on Linux using RPM packages:

  ```
  export ICE_HOME=/usr/share/Ice-3.5.0
  ```

- MacOS with homebrew:

  ```
  export ICE_HOME=/usr/local/Cellar/ice/3.5.0
  ```

- Windows using Visual Studio:

  ```
  set ICE_HOME=C:Ice-3.5.0
  ```

Users of a package manager will also need to set `ICE_HOME` if the Ice paths are not automatically detected correctly. The **slice2xxx** tools generally don't pick up the location of the slice definitions by default if this is unset.

Note: If the Ice headers and libraries are not present on the standard search paths, these will need to be specified using the `CPPPATH` and `LIBPATH` environment variables (see below).

Windows users building with Visual Studio will also need to run the Visual Studio environment setup scripts:

```
C:\Documents and Settings\USER>c:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\vcvarsall.bat
Setting environment for using Microsoft Visual Studio 2008 x86 tools.
```

or otherwise guarantee that the your environment is properly configured. For the 64bit build, be sure to use the right setup, namely *Start→Microsoft Visual Studio 2008→Visual Studio Tools→Visual Studio +2008 Command Prompt*.

12.5.4 Building the library

To build the C++ dynamic library:

```
cd omero
./build.py
./build.py build-cpp
```

or

```
./build.py build-all
```

If you would like to build the C++ tests, you can run:

```
./build.py test-compile-all
./build.py test-unit
```

or to test only C++:

```
./build.py -f components/tools/OmeroCpp/build.xml test
```

Note: If you would like to work on just the C++ code without worrying about the rest of the build, you can install **scons** and use it directly. Alternatively, you can use the **scons** version which comes with the OMERO source code:

```
cd components/tools/OmeroCpp && python ../../../target/scons/scons.py test
```

This **does** require having run the top-level build (`build.py`) at least once.

Note: If the build fails with errors such as

```
Checking for C++ header file Ice/Ice.h... no
Fatal Error: Ice/Ice.h not found
```

this can be caused by the Ice headers not being installed or not being on the search path. However, also check `components/tools/OmeroCpp/config.log`. If this contains error messages 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.

12.5.5 Further build configuration

The C++ bindings use scons[109] as a build system. **scons** provides several hooks into its operation. The following environment variables as defined in components/blitz/blitz_tools.py[110] are considered:

ARCH Either `x86` or `x64`. `x64` will be used by default on a 64-bit machine, otherwise `x86`

[109] http://www.scons.org
[110] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/blitz_tools.py

CPPPATH directories to be searched for include files, for example

`-I/opt/Ice-3.5.0/include`

A `:` or `;` separator character is used to separate directories, depending on the platform.

CXXFLAGS standard **make**-like CXXFLAGS variable

CXX compiler executable. Useful with ccache[111].

LIBPATH directories to be searched for libraries, for example

`-L/opt/Ice-3.5.0/lib`

Directories are separated by `:` or `;` as with CPPPATH.

ICE_HOME your Ice installation. The contained `include` directory will be added to your `CPPPATH`, but the contained `lib` directory will *not* be added to your `LIBPATH` since this may not contain the needed 32- or 64-bit libraries; this will need setting with the correct value for your system.

J specifies the number of concurrent build tasks as with **make**.

RELEASE `debug` or `Os` (i.e. optimize for size). `debug` is used by default.

VERBOSE show the actual build commands rather than the pretty "Compiling XYZ…" statements.

Zip files containing the C++ header files, the libraries, and source code are placed under `OMERO_HOME/target` with other zip artifacts.

Note: If you are using **make**, you can unpack the main zip (e.g. `OMERO.cpp-<version>-64dbg.zip`) to some directory (`OMERO_DIST`) and follow the instructions below to get started. For help with other build systems, please contact the mailing list.

12.5.6 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:

```
#
# MAKEFILE:
#
# Where the OMERO distribution was installed
OMERO_DIST?=/opt/omero

# Where the Ice lib/ and include/ directories are to be found
ICE_HOME?=/usr

INCLUDES=-I$(OMERO_DIST)/include -I$(ICE_HOME)/include

LIBS=-L$(OMERO_DIST)/lib -L$(ICE_HOME)/lib -L$(ICE_HOME)/lib64 \
     -lIce -lIceUtil -lGlacier2 -lomero_client -lstdc++

LIBPATH=$(LD_LIBRARY_PATH):$(ICE_HOME)/lib:$(ICE_HOME)/lib64:$(OMERO_DIST)/lib

.PHONY: clean run

yourcode.o: yourcode.cpp
        $(CXX) $(CXXFLAGS) -c -o $@ $< $(INCLUDES)

yourcode: yourcode.o
        $(CXX) -o $@ $^ $(LIBS)

run:  yourcode
        LD_LIBRARY_PATH="$(LIBPATH)" ./yourcode --Ice.Config=../etc/ice.config
```

[111] http://ccache.samba.org/

```
27
28 clean:
29         rm -f yourcode *.o *~ core
```

12.5.7 A trivial example: `yourcode.cpp`

And a simple example file might looking 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>
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*.

12.5.8 Compiling and running your code

Therefore, to compile and run **yourcode**, you will need to download the two files above (`Makefile` and `yourcode.cpp`) and then from the 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
```

12.5.9 Notes for Mac users

This example explains how to build on Linux only. For doing the same on Mac OS X, change all instances of `LD_LIBRARY_PATH` to `DYLD_LIBRARY_PATH`.

12.5.10 Notes for Visual Studio users

The `SConstruct` build file in *OMERO C++ language bindings* defines a target `msproj` which can be used to generate an MS Visual Studio project and solution. There is also a similarly named **ant** target:

```
build -f components\tools\OmeroCpp\build.xml msproj
```

Note: It may be necessary to specify `/Zm1000` as an additional compiler setting.

12.5.11 Further information

For the details behind writing, configuring, and executing a client, please see *Working with OMERO*.

See also:

Ice[112], *OMERO.grid*, *OMERO Application Programming Interface*, *Build System*, #1596[113] which added 64bit support

[112] http://www.zeroc.com
[113] http://trac.openmicroscopy.org.uk/ome/ticket/1596

CHAPTER

THIRTEEN

ANALYSIS

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

13.1 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[1].
- 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*.

13.2 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 *ExtendingOmero*).

For larger volumes of data, or data which needs to be queried, *OMERO.tables* provides a unified solution for the storage of columnar data from various sources, such as automated analysis results or script-based processing, and makes them available within OMERO.

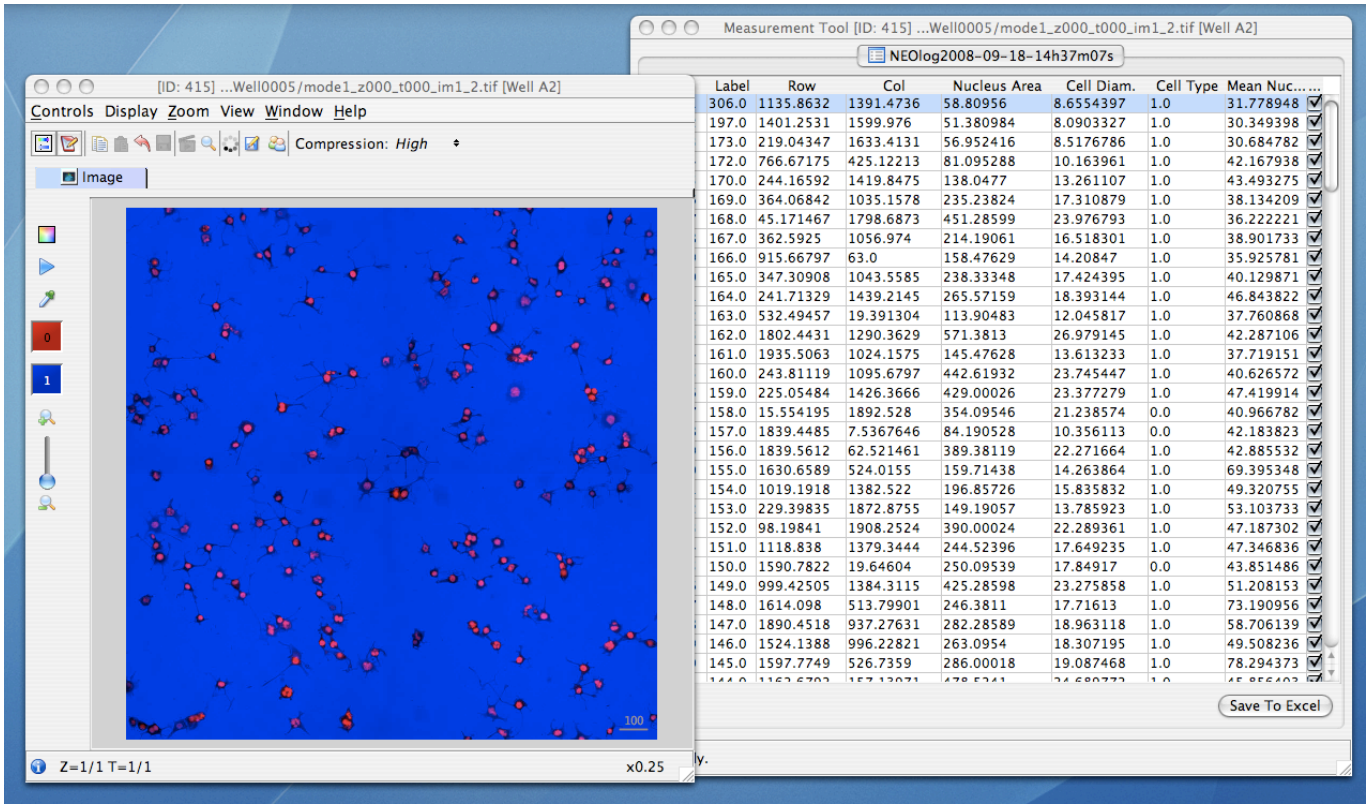
13.2.1 Third-party analysis and OMERO.tables

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[2]). In the Flex case, additional configuration may be necessary for accessing both the raw data and the analysis results. Watch the configuration screencast[3] for more information.

[1] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/
[2] http://cvs.openmicroscopy.org.uk/snapshots/movies/omero-4-1/mov/FlexPreview4.1-import.mov
[3] http://cvs.openmicroscopy.org.uk/snapshots/movies/omero-4-1/mov/FlexPreview4.1-configuration.mov

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 overlayed on the associated image:



13.2.2 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.[4] depending on your needs.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

13.3 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.

For installation instructions, see *Installing OMERO.tables*

13.3.1 The interface

The slice definition file[5] 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*.

[4] http://www.glencoesoftware.com/
[5] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/resources/omero/Tables.ice

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[6] 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.

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** (*[]*) – 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[]*) – 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.

[6] http://www.zeroc.com

- **size** (*long*) – The width of the array, >= *1*
- **values** (*[][]*) – 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 indicies 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 indicies to be retrieved from the table.

Returns The requested rows as a `Data` object.

`read`(*colNumbers*, *start*, *stop*)
Read a subset of columns and rows from a table.

Parameters

- **colNumber** (*long[]*) – A list of column indicies to be retrieved from the table.
- **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.

`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.

You many find the *Python* and *Java* annotated code samples helpful, in addition to the *examples* and documentation on the API[7]. These are only an introduction to using OMERO.tables and do not show its full potential, see *Going forward* for some inspiration.

13.3.2 Examples

- Hello World: examples/OmeroTables/first.py[8]
- Creating a Measurement Table: examples/OmeroTables/MeasurementTable.java[9]
- Querying a Table: examples/OmeroTables/FindMeasurements.java[10]

13.3.3 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.

13.3.4 Query language

The query language mentioned above is *currently* the PyTables condition syntax[11]. 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.

[7] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/grid/Table.html
[8] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroTables/first.py
[9] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroTables/MeasurementTable.java
[10] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroTables/FindMeasurements.java
[11] http://pytables.github.com/usersguide/condition_syntax.html

- `{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>x)', 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*.

13.3.5 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[12] 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[13]

For an example of production code that parses out such measurements, see populate_roi.py[14].

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*.

See also:

PyTables[15] Software on which OMERO.tables is built.

Condition Syntax[16] The PyTables condition syntax.

Tables.ice[17] The API definition for OMERO.tables

The Tables test suite[18] The testsuite for OMERO.tables

Installing OMERO.tables Installation requirements for install OMERO.tables

[12] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IRoi.html
[13] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroTables/iroi.py
[14] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroPy/src/omero/util/populate_roi.py

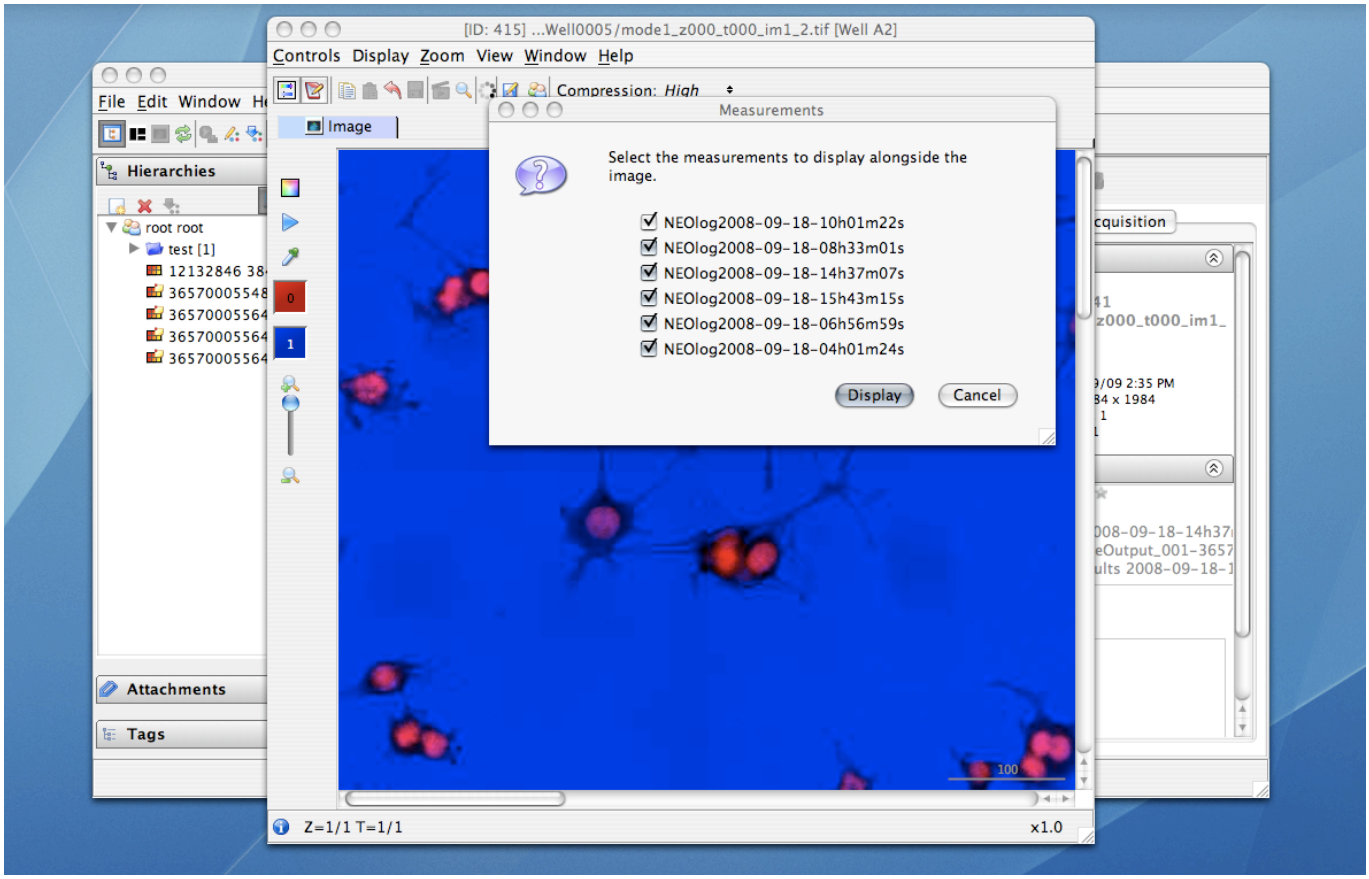


Figure 13.1: Choice between multiple measurements

CHAPTER

FOURTEEN

SCRIPTS - PLUGINS FOR OMERO

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

14.1 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 MATLAB scripts are also supported (currently using a Python wrapper as described in *MATLAB and scripting*, but native support will be available in OMERO 5). 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.

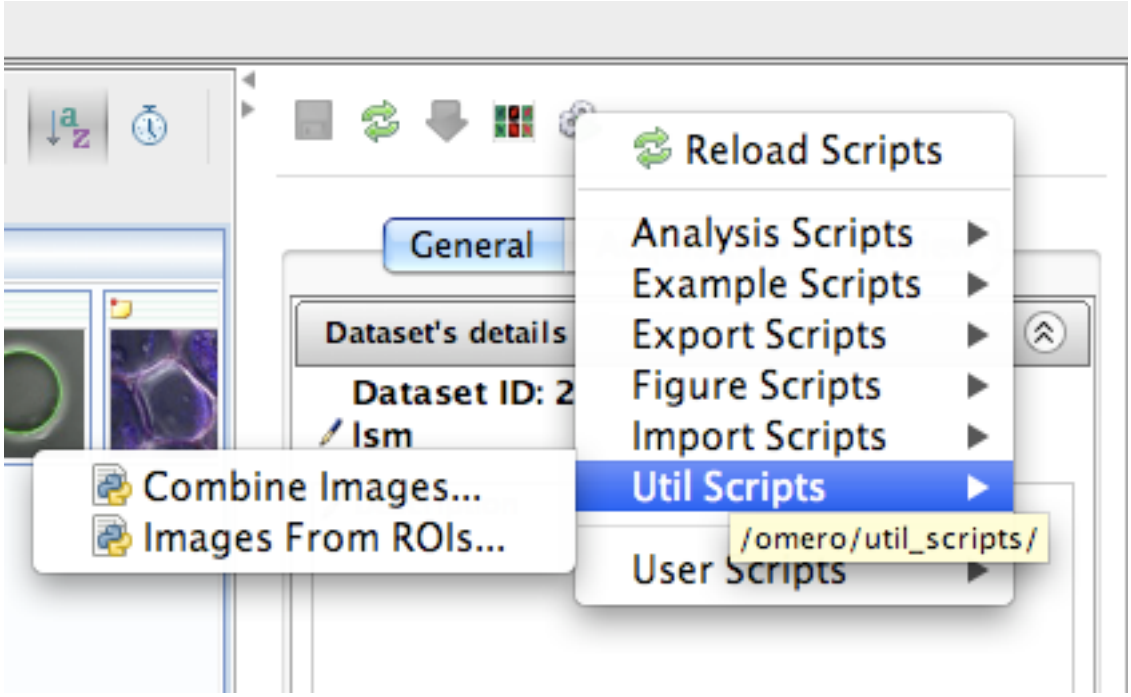


Figure 14.1: Scripts menu in OMERO.insight

14.1.1 Finding scripts

Core scripts[1] are bundled with every OMERO.server release and automatically available to all users. You can find additional scripts via the new script sharing[2] page.

[1] https://github.com/ome/scripts
[2] http://www.openmicroscopy.org/site/community/scripts

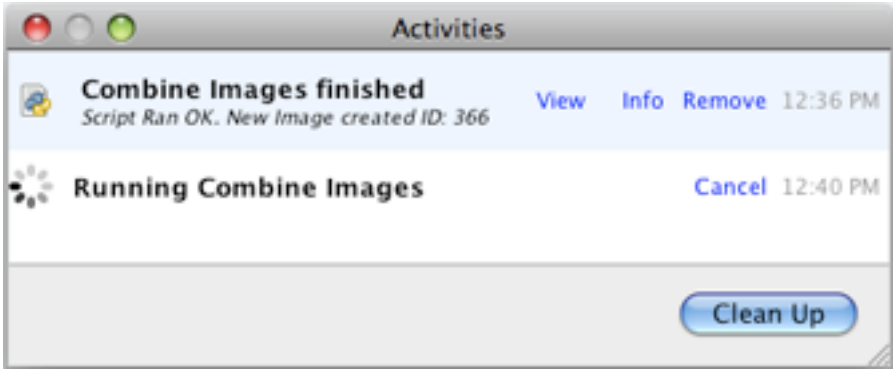


Figure 14.2: Running a script from an OMERO client

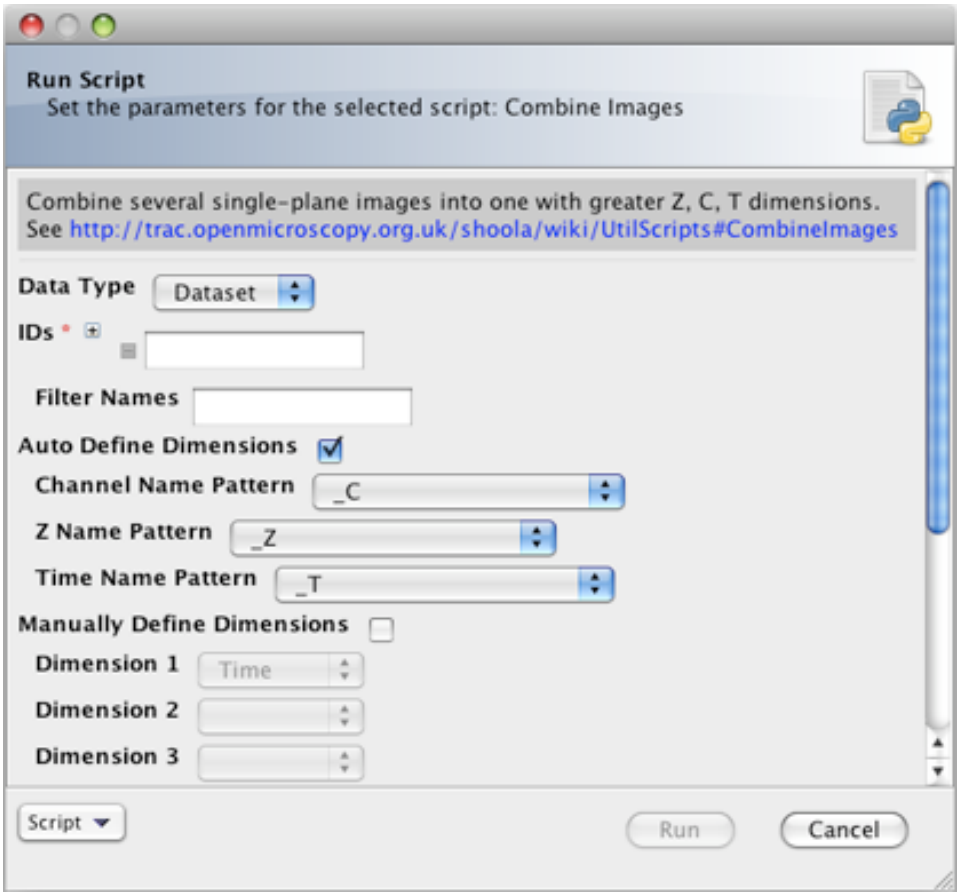


Figure 14.3: A script user interface

14.1.2 Installing and running 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.

14.1.3 Writing scripts

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.

If you are a biologist with no previous coding experience, you may find the Python for Biologists[3] free online course helpful.

[3] http://pythonforbiologists.com/index.php/introduction-to-python-for-biologists/

14.1.4 Managing scripts

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[4], you can use the GitHub app for your OS[5] (available for Mac and Windows but not Linux). The basic workflow is:

- fork our omero-user-script[6] 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

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. This means you should try to pick unique names to avoid future clashes e.g. Custom_Scripts/Search_Scripts/original_metadata_search.py:

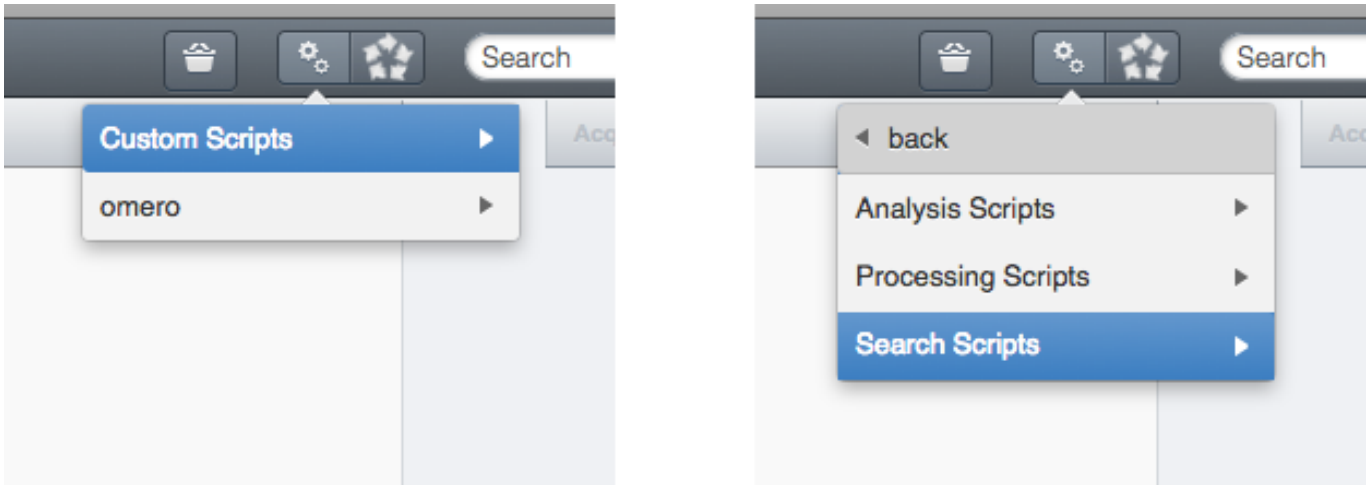


Figure 14.4: Custom scripts in OMERO.web menu

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.

14.1.5 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. We can either add your repository to the list on the script sharing[7] page so people can find it, or 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 scripting*

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

[4] http://www.openmicroscopy.org/site/support/contributing/using-git.html
[5] http://help.github.com/articles/set-up-git
[6] https://github.com/ome/omero-user-scripts
[7] http://www.openmicroscopy.org/site/community/scripts

14.2 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 script

'''
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 if for input, output or input
and output.
'''
client = script.client("SCRIPTNAME", "SCRIPTDESCRIPTION",
         script.TYPE("VARIABLENAME").[in()|out()|inout()], …)
# create a session on the server.
client.createSession()

# 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*.

14.2.1 Sample scripts

Below are two sample scripts. You can find the core scripts that are distributed with the OMERO.server under the scripts repository[8] or download them from OMERO.insight (from the bottom-left of any run-script dialog), or use the script sharing[9] page to find scripts written by other users.

Ping script

This script echoes the input parameters as outputs.

```
import omero, omero.scripts as script
client = script.client("ping.py", "simple ping script",
         script.Long("a"), script.String("b"))
client.createSession()

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.

[8] https://github.com/ome/scripts
[9] http://www.openmicroscopy.org/site/community/scripts

```
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
    client.setOutput("Channel%s" % i, wrap(str(ch.getLabel())))

# Cleanup
client.closeSession()
```

14.2.2 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 script to server
- Use command line (or OMERO.insight or web clients) to run script on the server (results will be displayed)
- Edit script and replace copy on 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[10]. 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 OMERO.insight (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 OMERO.insight (as discussed on the *Guidelines for writing OMERO.scripts*).

[10] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/ScriptingService/Edit_Descriptions.py

The script is well documented and should get you started. A few points to note:

Since the OMERO 4.3 release, if you are using the 'Blitz Gateway', 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 OMERO.insight client (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.

You may find it useful to add the OMERO.server/bin/ folder to your PATH so you can call `bin/omero` commands when working in the scripts folder. E.g:

```
export PATH=$PATH:/Users/will/Desktop/OMERO.server-4.4.12/bin/
```

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. Cur
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 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.

```
wjm:examples will$ omero script launch 301  # script ID
Using session 1202acc0-4424-4fa2-84fe-7c9e069d3563 (root@localhost:4064). Idle timeout: 10.0 min. Curre
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 sc
```

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: /Users/will/Documents/workspace/Omero/dist/bin/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 /Users/will/Documents/workspace/Omero/dist/bin/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. Curre
 id  | Official scripts
-----+-----------------------------------------
 201 | /omero/analysis_scripts/flim-omero.py
 1   | /omero/analysis_scripts/FLIM.py
 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
 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
 9   | /omero/setup_scripts/FLIM_initialise.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.

```
$ wjm:examples will$ omero script params 301
Using session 1202acc0-4424-4fa2-84fe-7c9e069d3563 (root@localhost:4064). Idle timeout: 10.0 min. Curre

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.
See http://www.openmicroscopy.org/site/support/omero4/developers/scripts/user-guide.html for the tutori
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:
```

14.2.3 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 OMERO.insight 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

14.3 Guidelines for writing OMERO.scripts

These guidelines for writing Python 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[11].

14.3.1 Script naming and file path

- Script Name should be in the form 'Script_Name.py'. OMERO.insight will replace underscores with spaces in the script selection menu.
- File paths - OMERO.insight 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 OMERO.insight 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.

[11] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/ScriptingService/Edit_Descriptions.py

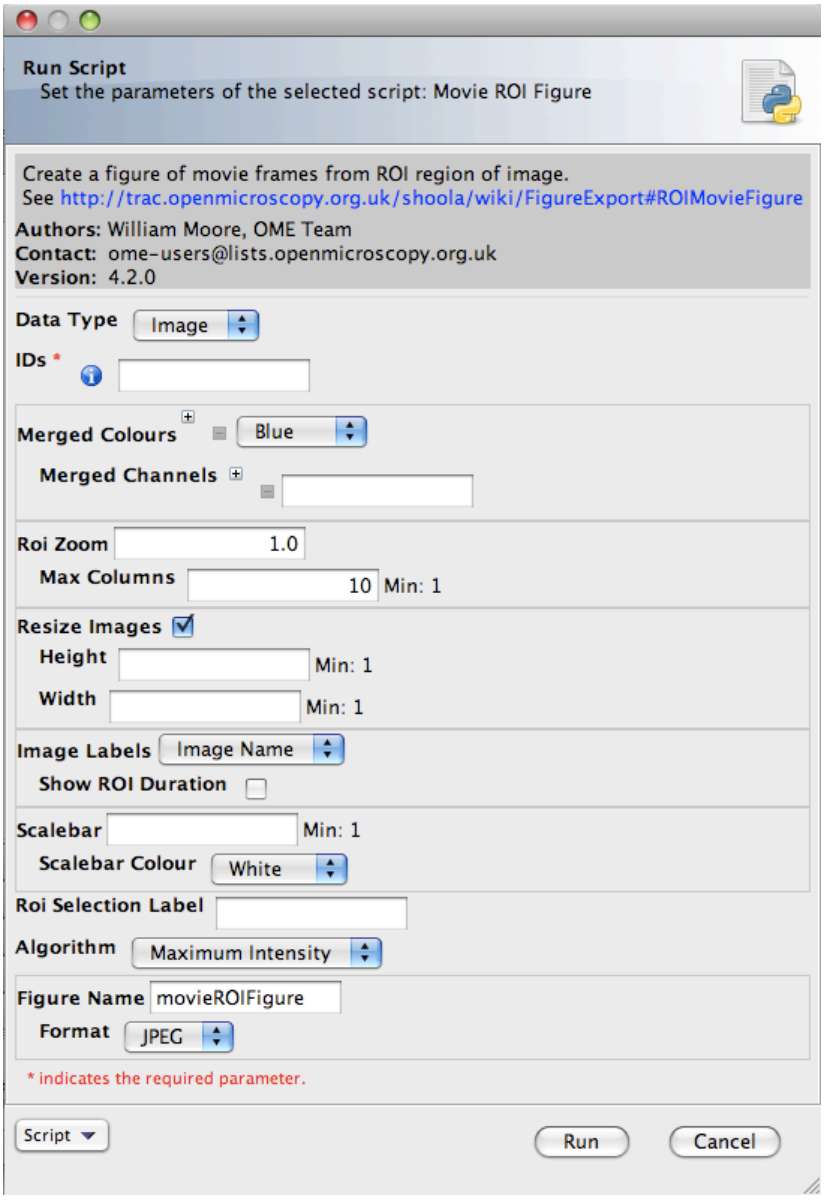


Figure 14.5: Movie ROI figure script UI

14.3.2 Parameters

- Parameter Names should be in the form 'Parameter_Name'. Underscores will be replaced by spaces in the UI generated in OMERO.insight.
- 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 OMERO.insight 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_E
```

- 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),
```

14.3.3 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 lexographic 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"),
```

14.3.4 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. E.g.

```
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))
    )
```

14.3.5 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 OMERO.insight when the script completes.

  ```
  client.setOutput("Message", rstring("Script generated new Image"))
  ```

- Scripts that generate an Image should return the ImageI object. OMERO.insight 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. OMERO.insight 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))
  ```

14.3.6 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.
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

14.4 MATLAB and scripting

The scripting service can also run MATLAB scripts. This is done using the python package Mlabwrap[12], which allows access to MATLAB functions from OMERO.blitz scripts.

14.4.1 Installing Mlabwrap

To install MlabWrap follow the installation guide at http://www.scipy.org/MlabWrap and make sure that the paths are set for the environment variables:

```
LD_LIBRARY_PATH=$MATLABROOT/bin/Platform
MLABRAW_CMD_STR=$MATLABROOT/bin/matlab
```

14.4.2 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, omero.scripts as script
# import mlabwrap to launch matlab.
from mlabwrap import matlab;
client = script.client("rand.py", "Get matrix of random numbers drawn from a uniform distribution",
                       script.Long("x").inout(), script.Long("y").inout())
client.createSession()

x = client.getInput("x").val
y  = client.getInput("y").val

# call the matlab rand function via mlabwrap will automatically launch matlab
# if it is not already running on the system and call the rand method.
val = matlab.rand(x,y);
print val
```

[12] http://mlabwrap.sourceforge.net

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.

```
import omero, omero.scripts as script
# import mlabwrap to launch matlab.
from mlabwrap import matlab;
client = script.client("projection.py", "Call the matlab projection code",
                        script.String("iceConfig").in(), script.String("user").in(),
                        script.String("password"),
                        script.Long("pixelsId").inout(), script.String("method").inout()
                        script.Long("stack").inout())
client.createSession()

iceConfig = client.getInput("pixelsId").val
user = client.getInput("pixelsId").val
password = client.getInput("pixelsId").val
method  = client.getInput("method").val
stack = client.getInput("stack").val;

image = matlab.performProjection(iceConfig, username, password, pixelsId, stack, method);
```

The MATLAB projection method

```
function performProjection(iceConfig, username, password, pixelsId, zSection, method)

omerojavaService = createOmeroJavaService(iceConfig, username, password);
pixels = getPixels(omerojavaService, pixelsId);
stack = getPlaneStack(omerojavaService, pixelsId, zSection);
projectedImage = ProjectionOnStack(stack, method);



function [resultImage] = ProjectionOnStack(imageStack,type)

[zSections, X, Y] = size(imageStack);

if(strcmp(type,'mean') || strcmp(type, 'sum'))
    resultImage = squeeze(sum(imageStack));
    if(strcmp(type,'mean'))
        resultImage = resultImage./zSections;
    end
end
if(strcmp(type,'max'))
    resultImage = squeeze(max(imageStack,[],1));
end
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

14.5 OMERO.scripts advanced topics

14.5.1 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).

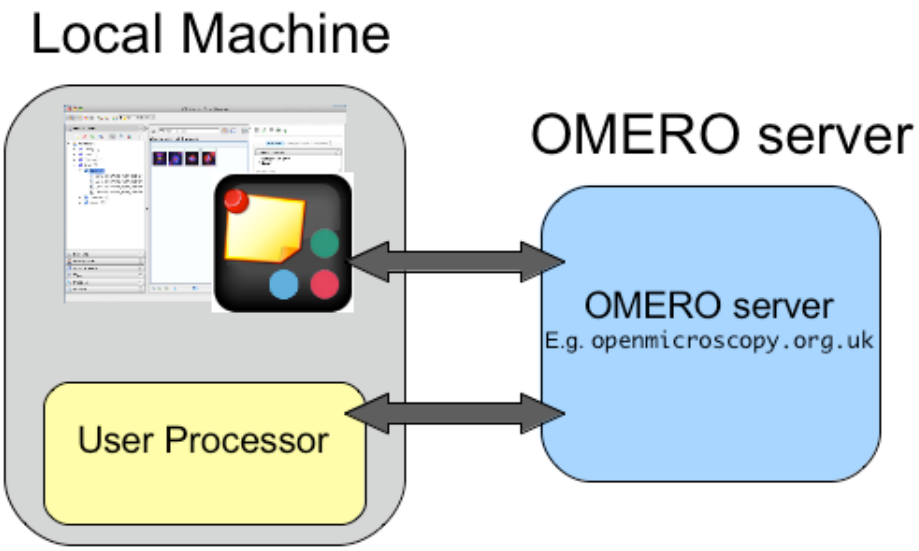


Figure 14.6: OMERO scripting workflow

- You need to download 'Ice' from ZeroC and set the environment variables, as described in the server installation page (see *Unix* and *Windows* versions).
- You also need the OMERO server download. Go to the OMERO downloads[13] page and get the appropriate server package (version must be OMERO 4.2 or later and 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-Beta-4.2/
$ bin/omero -s openmicroscopy.org.uk -u will script serve user
$ password: ......
```

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-Beta-4.2/
$ bin/omero -s openmicroscopy.org.uk -u will user list
$ bin/omero -s openmicroscopy.org.uk -u will 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 -s openmicroscopy -u will script list user      # lists user scripts
 id  | Scripts for user
    -----+-------------------------------------------------------------------------------------------
 151 | examples/HelloWorld.py
 251 | examples/Edit_Descriptions.py
```

[13] http://www.openmicroscopy.org/site/products/omero/downloads

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.

14.5.2 The iScript service

The OMERO.blitz server provides a service called iScript[14] 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 `bin/omero script` or the OMERO.insight client to work with scripts as described on the *OMERO.scripts user guide*.

14.5.3 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 examples/ScriptingService/adminWorkflow.py[15]. 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 examples/ScriptingService/adminWorkflow.py[16], 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
```

[14] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IScript.html
[15] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/ScriptingService/adminWorkflow.py
[16] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/ScriptingService/adminWorkflow.py

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 examples/ScriptingService/runHelloWorld.py[17]. 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"),
   "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['Output_Message'].getValue()
if 'stdout' in results:
    origFile = results['stdout'].getValue()
    print "Script generated StdOut in file:" , origFile.getId().getValue()
if 'stderr' in results:
```

[17] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/ScriptingService/runHelloWorld.py

```
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.

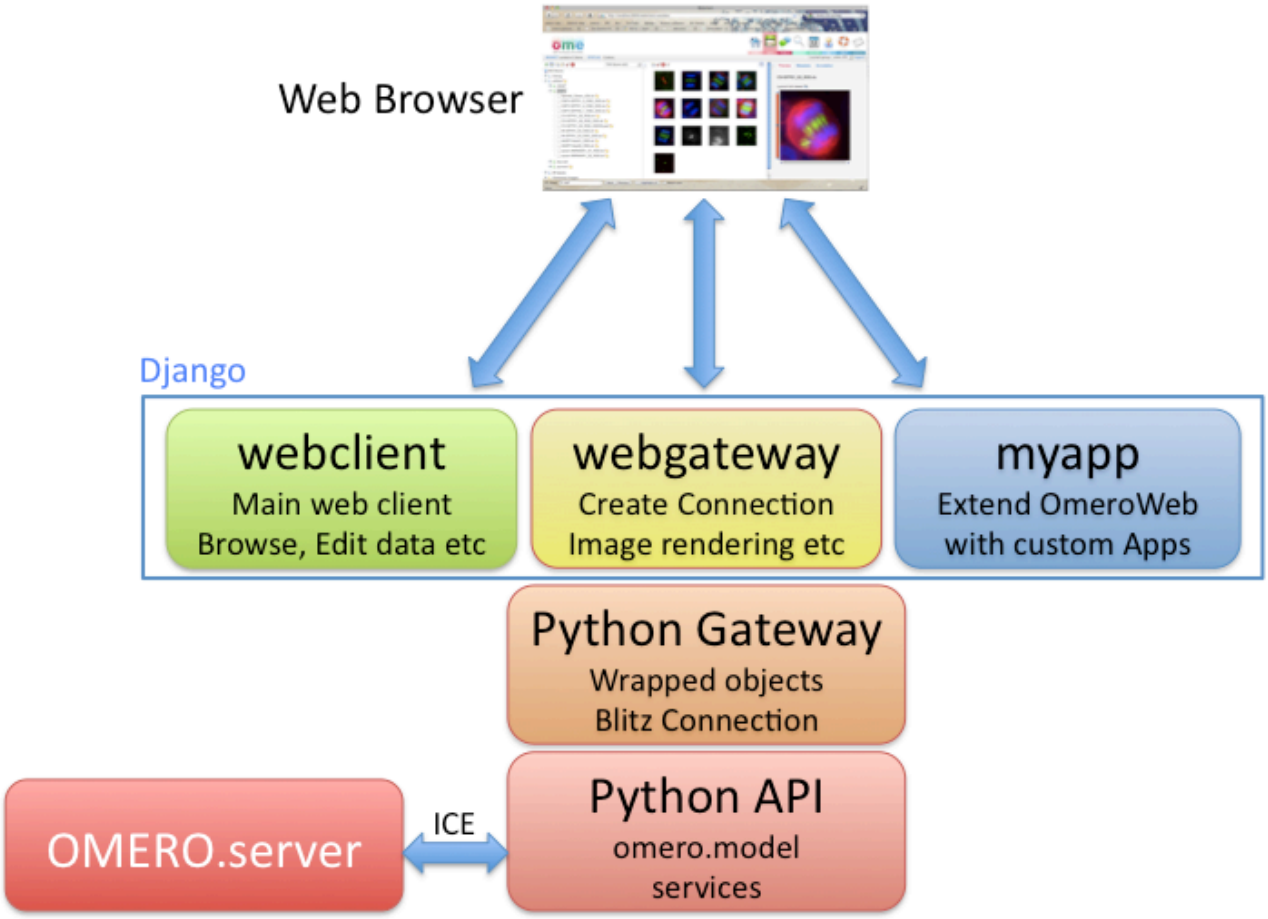
CHAPTER

FIFTEEN

WEB

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

15.1 OMERO.web framework



MOVIE introduction to OmeroWeb[1]

OMERO.web is a framework for building web applications for OMERO. It uses Django[2] to generate HTML pages 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

[1] http://cvs.openmicroscopy.org.uk/snapshots/movies/omero-4-3/mov/OmeroWebIntro-4.3.mov
[2] https://www.djangoproject.com/

webclient (working with image data) and webadmin (administrator management). This modular framework makes it possible to extend OMERO.web with your own apps.

One of the apps, *WebGateway*, provides utility methods for accessing data and rendering images, as well as handling the connection to OMERO server.

15.1.1 OMERO.web infrastructure

OMERO Python API

The OMERO.web framework is all based on the OMERO Python API, using the Blitz Gateway (see *OMERO Python language bindings*). The OMERO.web framework provides functionality for creating and retrieving connections to OMERO (see example below & *Writing OMERO.web views* for more details)

Web gateway

The webgateway is a Django app that provides utility functionality for the other web components. This includes a full image viewer, as well as methods for rendering images etc. You can browse the webgateway URLs[3], or see the *WebGateway* page for more info.

Web apps

The OMERO.web framework consists of several Django apps denoted by folders named 'web....'. These include webgateway & webtest, as discussed above, as well as released tools (webadmin, webclient) and other apps in development:

- webclient: Main web client for viewing images, annotating etc. More information available under *OMERO.web*.
- webadmin: For administration of user and group settings.
- webgateway: A web services interface, providing rendered images and data. See *WebGateway*.
- webtest: A sample app for testing, that can also be used as a basis for creating your own app.

15.1.2 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 - 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 'webtest' app. Also, see the OMERO.web demo movie[4].

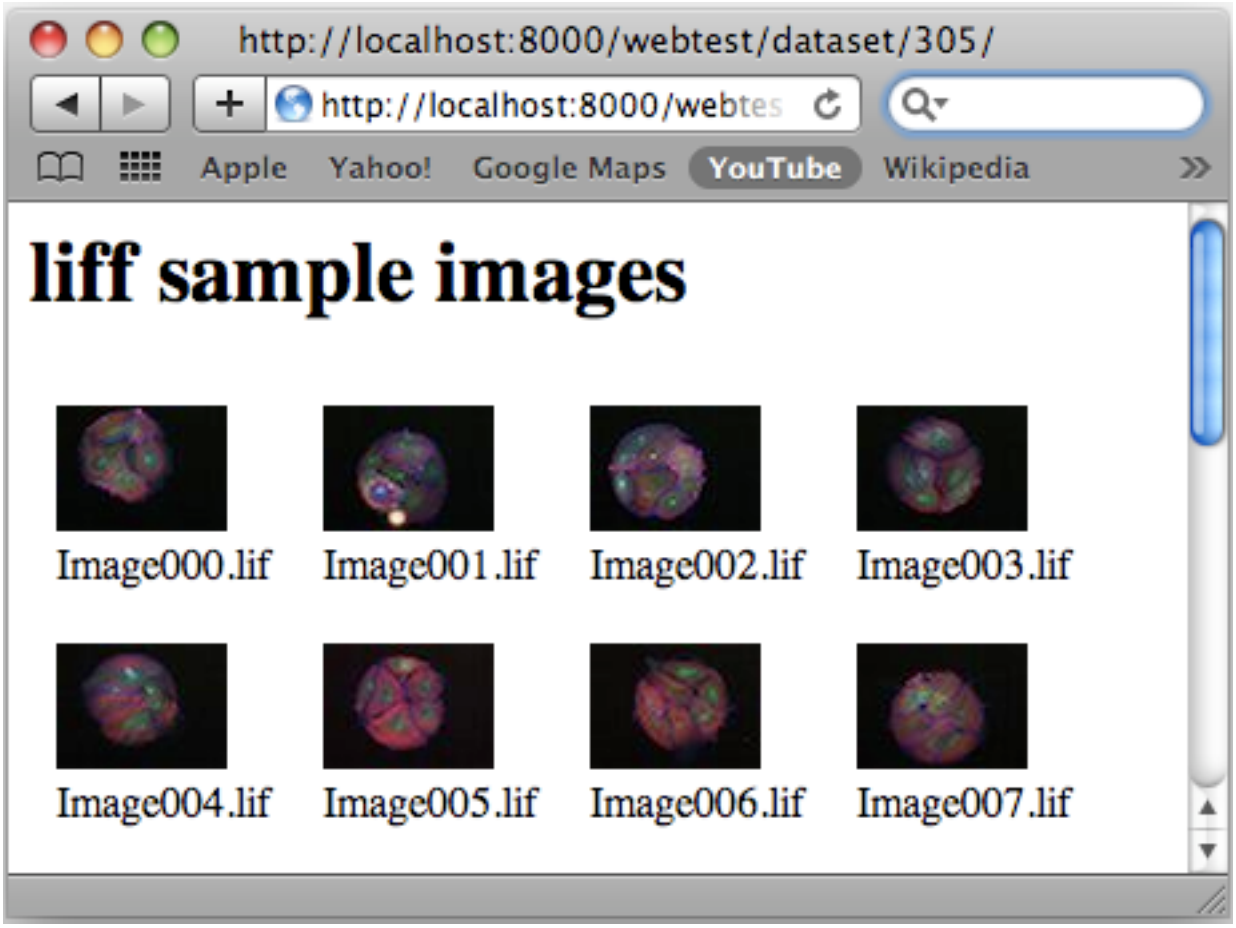
Note: Some details of this code have changed in the OMERO 4.4 release - see below.

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, you can view the page under *http://servername.example.org/webtest/dataset/<datasetId>*.

- **url** goes in omeroweb/webtest/urls.py This maps the URL 'webtest/dataset/<datasetId>/' to the View function 'dataset', passing it the datasetId.

```
url( r'^dataset/(?P<datasetId>[0-9]+)/$', views.dataset ),
```

[3] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omeroweb.webgateway.urls-module.html
[4] http://cvs.openmicroscopy.org.uk/snapshots/movies/omero-4-3/mov/OmeroWebIntro-4.3.mov



- **View** function, in omeroweb/webtest/views.py. NB: @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). Was @isUserConnected before OMERO 4.4
@login_required()
def dataset(request, datasetId, conn=None, **kwargs):
    ds = conn.getObject("Dataset", datasetId)
    # generate html from template
    return render_to_response('webtest/dataset.html', {'dataset': ds})
```

- **Template:** The template web page, in omeroweb/webtest/templates/webtest/dataset.html

```
<html><body>

<h1>{{ dataset.getName }}</h1>

{% for i in dataset.listChildren %}
    <div style="float:left; padding:10px">
        <img src="{% url webgateway.views.render_thumbnail i.id %}" />
        <br />
        {{ i.getName }}
    </div>
{% endfor %}

</body></html>
```

- Next: Get started by *Creating an app*....

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to

http://openmicroscopy.org/site/support/omero/

15.2 Creating an app

The Django web site has a very good tutorial[5] 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. One major feature of Django that we do not use in OmeroWeb is the Django database mapping, since all data comes from the OMERO server and is saved back there. You will notice that the models.py files in each app are empty.

15.2.1 Getting set up

You will need to have an OMERO server running that you can connect to. This will typically be on your own machine, although it does not necessarily have to be. If you want to connect to another server (not localhost) you can edit the server list as described on the OMERO.web deployment page (see *Unix* or *Windows* version) and choose that server when you log in. That page also describes how to set debug to 'True' which is important when developing with OMERO.web and you should also be using the Django 'development' server.

```
$ bin/omero web start
Starting django development webserver...
Validating models...
0 errors found

Django version 1.1.1, using settings 'omeroweb.settings'
Development server is running at http://0.0.0.0:4080/
Quit the server with CONTROL-C.
```

Note: Port number is 4080

You should make sure that you can access the webclient and webadmin on your local machine before starting to develop your own code. Be sure to use the correct port number, E.g:

- http://localhost:4080/webclient/[6]

When you edit and save your app, Django will automatically detect this and you only need to refresh your browser to see the changes. You can see this in the OMERO.web intro movie[7].

If you want to run OMERO.web from source code, see *Editing OMERO.web*.

You can place your app anywhere on your `PYTHONPATH`, as long as it can be imported by OMERO.web.

15.2.2 Creating an app

We suggest you use github (as we do) since it is much easier for us to help you with any problems you have if we can see your code. The steps below describe how to create a stand-alone git repository for your app, similar to webtagging[8]. If you do not want to use github, simply ignore the github steps below.

The steps below describe setting up a new app. You should choose an appropriate name for your app and use it in place of <your-app> in the examples below:

Create and checkout a new github repository OR manually create a new directory

- Login to your github account homepage (E.g. https://github.com/<your-name>/) and click "New repository"
- Enter the name of <your-app>, add description and choose to add README.

[5] https://docs.djangoproject.com/en/dev/intro/tutorial01/
[6] http://localhost:4080/webclient/
[7] http://cvs.openmicroscopy.org.uk/snapshots/movies/omero-4-3/mov/OmeroWebIntro-4.3.mov
[8] https://github.com/dpwrussell/webtagging

- Checkout your new repository (into a new directory)

```
$ git clone git@github.com:<your-name>/<your-app>.git
```

- OR: If you haven't used git to create your app directory above, then

```
$ mkdir <your-app>
```

Add your app location to your PYTHONPATH

If your app is not in a directory that is already on your `PYTHONPATH` then you need to add it:

```
$ export PYTHONPATH=$PYTHONPATH:/path/to/your-app
```

Add the essential files to your app

- Create an empty file `<your-app>/__init__.py` (NB: both double underscores)
- Create urls.py

```
from django.conf.urls.defaults import *
from omeroweb.<your-app> import views

urlpatterns = patterns('django.views.generic.simple',

     # index 'home page' of the <your-app> app
     url( r'^$', views.index, name='<your-app>_index' ),

 )
```

- Create views.py

```
from django.http import HttpResponse

def index(request):
    """
    Just a place-holder while we get started
    """
    return HttpResponse("Welcome to your app home-page!")
```

Add your app to OMERO.web

This will add your app to the INSTALLED_APPS, so that URLs are registered etc.

```
$ bin/omero config set omero.web.apps '["<your-app>"]'
```

Note: For releases before 4.4, you need to 'register' your app with Django manually by adding it to the INSTALLED_APPS list in omeroweb/settings.py following the pattern of existing apps there. You also need to edit omeroweb/urls.py to add your app's urls.py file to the list of "urlpatterns". Again, you should be able to follow the existing examples there. You can also specify at this point the URL under which your app will be found.

Now you can view the home-page we created above (NB: you will need to restart the OMERO.web server for the config settings to take effect)

```
$ bin/omero web stop
$ bin/omero web start
```

Go to http://localhost:4080/[9]<your-app>/ OR http://localhost:8000/[10]<your-app>/ Should see 'Welcome'

Commit your code and push to github

```
$ git status (see new files, plus .pyc files)
$ echo "*.pyc" > .gitignore          # ignore .pyc files
$ echo ".gitignore" >> .gitignore    # ALSO ignore .gitignore

$ git add ./
$ git commit -m "Initial commit of bare-bones OMERO.web app"
$ git push origin master
```

Connect to OMERO: example

We have got our new app working, but it is not connecting to OMERO yet. Let us create a simple "stack preview" for an Image with multiple Z-sections. We are going to display the image name and 5 planes evenly spaced across the Z-stack. You should be able to add the appropriate code to urls.py, views.py that you created above, and add a template under /omeroweb/<your-app>/templates/<your-app>/

Note: note that /<your-app>/ appears twice in that path (need an extra folder under templates). This example can be found in webtest.

- **urls.py**

```
url( r'^stack_preview/(?P<imageId>[0-9]+)/$', views.stack_preview,
     name="<your-app>_stack_preview" ),
```

- **views.py** Here we are using the @login_required decorator to retrieve a connection to OMERO from the session key in the HTTP request (or provide a login page and redirect here). 'conn' is passed to the method arguments. NB: Note a couple of new imports to add at the top of your page.

```
from omeroweb.webclient.decorators import login_required
from django.shortcuts import render_to_response


@login_required()
def stack_preview (request, imageId, conn=None, **kwargs):
    """ Shows a subset of Z-planes for an image """
    image = conn.getObject("Image", imageId)        # Get Image from OMERO
    image_name = image.getName()
    sizeZ = image.getSizeZ()                         # get the Z size
    # 5 Z-planes
    z_indexes = [0, int(sizeZ*0.25),
              int(sizeZ*0.5), int(sizeZ*0.75), sizeZ-1]
    return render_to_response('webtest/stack_preview.html',
         {'imageId':imageId,
         'image_name':image_name,
         'z_indexes':z_indexes})
```

- **<your-app>/templates/<your-app>/stack_preview.html**

[9] http://localhost:4080/
[10] http://localhost:8000/

```
<html>
<head>
     <title>Stack Preview</title>
</head>
<body>
     <h1>{{ image_name }}</h1>

     {% for z in z_indexes %}
         <img src="{% url webgateway.views.render_image imageId z 0 %}"
             style="max-width: 200px; max-height:200px"/>
     {% endfor %}
</body>
</html>
```

Viewing the page at http://localhost:4080/<your-app>/stack_preview/<image-id>/ should give you the image name and 5 planes from the Z stack. You will notice that we are using webgateway to handle the image rendering using a URL auto-generated by Django - see *WebGateway*.

Resources for writing your own code

The webtest app has a number of examples. If you go to the webtest homepage E.g. http://localhost:8000/webtest you will see an introduction to some of them. This page tries to find random image and dataset from your OMERO server to use in the webtest examples.

Extending templates

We provide several HTML templates in webgateway/templates/webgateway/base. This is a nice way of giving users the feeling that they have not left the webclient, if you are providing additional functionality for webclient users. You may choose not to use this if you are building a 'stand-alone' web application. In either case, it is good practice to create your own templates with common components (links, logout etc), so you can make changes to all your pages at once. See *Writing page templates in OMERO.web* for more info.

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 in omeroweb/settings.py. The list of CUSTOM_SETTINGS_MAPPINGS in omeroweb/settings.py code 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 yourapp.foo, that contains a dictionary of key-value pairs, you can add to CUSTOM_SETTINGS_MAPPINGS in yourapp/settings.py:

```
import json
CUSTOM_SETTINGS_MAPPINGS = {
    "omero.web.yourapp.foo": ["FOO", '{"key": "val"}', json.loads]
}
```

From somewhere else in your app, you can then access the settings:

```
from yourapp import settings

print settings.FOO
```

Users can then configure this on the command line as follows:

```
$ bin/omero config set omero.web.yourapp.foo '{"userkey": "userval"}'
```

OMERO.web top links

You can configure settings 'top_links' to add a link 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 urls.py.

```
url( r'^$', views.index, name='webmobile_index' ),
```

- **Update configuration** Use the OMERO command line interface to add the link or links to the appropriate list. NB: Since there is not currently an option to **add** to web settings lists, you will need to include the full list of links when you configure the list.

To add a single link, using the format ["Label", "URL_name"], you can follow this example:

```
$ bin/omero config set omero.web.ui.top_links '[["Mobile", "webmobile_index"]]'
```

Multiple links can be added in the same way. You can also create **external** links by specifying the full URL instead of the "URL_name". For example:

```
$ bin/omero config set omero.web.ui.top_links '[["Mobile", "webmobile_index"], ["OME", "https://www.open
```

OMERO.web plugins (OMERO 4.4)

If you want to display content from your app within the webclient UI, please see *Webclient Plugins*.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

15.3 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 app[11] and there are also some examples in the webtest app.

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.

15.3.1 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.

The webclient plugins work by adding some custom Javascript snippets into the main pages of the weblient 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'll 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.

[11] http://www.openmicroscopy.org/site/products/partner/omero.webtagging

15.3.2 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 ),
```

These URLs should simply display the content that you want to show in the webclient. NB: 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. Furthermore, it is important not to overwrite CSS or Javascript in the webclient (for example by including jQuery).

15.3.3 Configuring the plugin

Choose an element ID

You will need to specify an ID for the * * 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/<app-name>/* directory and named *.html*, e.g. /templates/<app-name>/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 option 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>
```

15.3.4 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.

Note: Since there is not currently an option to **add** to web settings lists, you will need to include the full list of plugins when you configure the plugin list.

The OMERO.webclient does not include any center plugins by default, so if you only want to add a single plugin to the center, you can simply do:

```
$ bin/omero config set omero.web.ui.center_plugins
    '[["Auto Tag", "webtagging/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 keep these and add your plugin to the list, you will need to list all three. For example, to add the webtest ROI plugin:

```
$ bin/omero config set omero.web.ui.right_plugins
    '[["Acquisition", "webclient/data/includes/right_plugin.acquisition.js.html", "metadata_tab"],
    ["Preview", "webclient/data/includes/right_plugin.preview.js.html", "preview_tab"],
    ["ROIs", "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). NB: 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.

15.3.5 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

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

15.4 Editing OMERO.web

If you need to make changes to OMERO.web itself, then you will want to check out the OMERO source code. You can directly edit and run the OMERO.web code. This means that you benefit from the convenience of editing, saving and refreshing your browser without any build step.

However, you will still need to build OMERO (or download the release build) and set up your `PYTHONPATH` as described in the install documentation in order that you have the various dependencies such as Django.

You will then have 2 copies of the OMERO.web code - source code under components/tools/OmeroWeb/omeroweb and the server build under dist/lib/python/omeroweb.

To set up and run OMERO.web from the source code, you need to follow a few steps (commands are shown below):

- Set $OMERO_HOME, so that OMERO.web knows where to find config, write logs etc.

  Note: You should not set $OMERO_HOME on production servers

- Make sure that the Django libraries that are under the build: dist/lib/python/django are on your `PYTHONPATH`.
- Remove the built omeroweb folder, otherwise this will get used instead of the source omeroweb

  Note: You have to do this again if you build the server

- From the source omeroweb/ folder, manually run the Django development server

```
# Example path to build target or downloaded directory
$ export OMERO_HOME = ~/Desktop/OMERO/dist

# Make sure the Django code etc can be imported
$ export PYTHONPATH=$OMERO_HOME/lib/python/:$PYTHONPATH
$ cd $OMERO_HOME
# need to remove the built omeroweb code so it doesn't get imported
$ rm -rf lib/python/omeroweb/

$ cd ../components/tools/OmeroWeb/omeroweb
$ python manage.py runserver
Validating models...

0 errors found
Django version 1.3.1, using settings 'omeroweb.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

  Note: Default port number is 8000. To specify port, use E.g: $ python manage.py runserver 0.0.0.0:4080

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

15.5 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.web user guides[12] for more details.

15.5.1 Web services

This list of URLs below may be incomplete or out of date. For a complete list of URLs, see the latest API, latest API[13] 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 webclient or webadmin login pages 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>/`

[12] http://help.openmicroscopy.org/
[13] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/epydoc/omeroweb.webgateway.urls-module.html

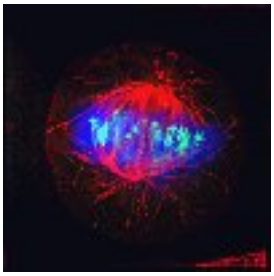


Figure 15.1: Rendered thumbnail

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.

```
webgateway/img_detail/<imageid>/
{% url webgateway.views.full_viewer image_id %}
```

Images

- Returns a jpeg of the specified plane with current rendering settings

```
webgateway/render_image/<imageid>/<z>/<t>/
{% url webgateway.views.render_image image_id theZ theT %}

From OMERO 4.4.4, omitting Z and T will use the default values:
webgateway/render_image/<imageid>/
{% url webgateway.views.render_image image_id %}
```

- Makes a jpeg laying out each active channel in a separate panel

```
webgateway/render_split_channel/<imageId>/<z>/<t>/
{% url webgateway.views.render_split_channel image_id theZ theT %}
```

- Plots the intensity of a row of pixels in an image. w is line width

```
webgateway/render_row_plot/<imageId>/<z>/<t>/<y>/<w>
{% url webgateway.views.render_row_plot image_id theZ theT yPos width %}
```

- Plots the intensity of a column of pixels in an image.

```
webgateway/render_col_plot/<imageId>/<z>/<t>/<x>/<w>/
{% url webgateway.views.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 75). Specify just one to retain aspect ratio

```
webgateway/render_thumbnail/<imageId>/<w>/<h>
{% url webgateway.views.render_thumbnail image_id 100 %}      # size 100
{% url webgateway.views.render_thumbnail image_id %}          # default size
```

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 a 4 channel image, to turn on all channels except 2:

```
?c=1,-2,3,4

# From OMERO 4.4.4 you can simply specify the active channels
?c=3          # only Channel 3 is active
?c=3,4        # Channels 3 and 4 are active
```

- Channel colour. E.g. To set the colours 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 a 3 Channel image.

```
?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     # OMERO 4.4.4 only: inactive channels can be omitted
```

- Z-projection. Maximum intensity, Mean intensity or None (normal)

```
?p=intmax
?p=intmean
?p=normal
```

- Rendering 'Mode': greyscale or colour.

```
?m=g    # greyscale (only the first active channel will be shown in grey)
?m=c    # colour
```

- Parameters can be combined, E.g.

```
webgateway/render_image/2602/10/0/?c=1|100:505$0000FF,2|463:2409$00FF00,3|620:3879$FF0000,-4|447:4
```

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>/`

Saving etc

- `webgateway/saveImgRDef/<imageId>/`
- `webgateway/resetImgRDef/<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,
    "z": 31,
    "t": 1,
    "height": 480
}
}
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

15.6 Embedding OMERO.web viewport to your website

Insert the following: **WARNING**: Please note that you are giving plain password and everyone can read from your HTML source code!

```
<div id="omeroviewport"><iframe width="850" height="600" src="http://localhost:8000/webclient/login/?us
```

15.6.1 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,menuba
        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 vie
```

15.6.2 Embedding OMERO.web viewport to the template in OMERO.web

This website will show you how to easily embed the viewport to the new template with the use of the jQuery JavaScript? library.

Use the following code to reference the stylesheets and scripts.

```
<link rel="stylesheet" href="/appmedia/webgateway/css/jquery-plugin-gs_slider.css" type="text/css" medi
<link rel="stylesheet" href="/appmedia/webgateway/css/weblitz-viewport.css" type="text/css" media="scre

<script type="text/javascript" src="/appmedia/omeroweb/javascript/jquery_1.3.2.js"></script>

<script type="text/javascript" src="/appmedia/webgateway/js/weblitz-viewport.js"></script>
<script type="text/javascript" src="/appmedia/webgateway/js/jquery-plugin-viewportImage.js"></script>
<script type="text/javascript" src="/appmedia/webgateway/js/jquery-plugin-gs_slider.js"></script>
<script type="text/javascript" src="/appmedia/webgateway/js/gs_utils.js"></script>
```

Then create the small java script which allows you to view particular image defined by **image_id**. Please note that if you also need to modify the name of your application **/my_app**you are running in.

```
<script type="text/javascript">
    $(document).ready(function()
        {
            var viewport = $.WeblitzViewport($("#viewport"), "/MYAPP" );
            viewport.load(image_id);

    });
</script>
```

Then in **<BODY>** insert the following:

```
<div class="miniview" id="viewport"></div>
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

15.7 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 web-gateway/views.py code. In general, these notes refer to the 4.4 and later release of OMERO, since the OMERO.web framework will be cleaned up substantially in this release. We note some points below where these changes will break existing code.

Although we aim to provide some useful notes and examples here, you will find the best source of examples is the code itself[14].

15.7.1 @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()

Note: Before 4.4, this was called @isUserConnected and had similar functionality.

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 webtest/views.py). Note the Blitz Gateway connection "conn" retrieved by @login_required() is passed to the function via the optional parameter `conn=None`.

[14] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroWeb/omeroweb/webclient/views.py

```
from omeroweb.decorators import login_required

@login_required()
def dataset(request, datasetId, conn=None, **kwargs):
    ds = conn.getObject("Dataset", datasetId)
    return render_to_response('webtest/dataset.html', {'dataset': ds})
```

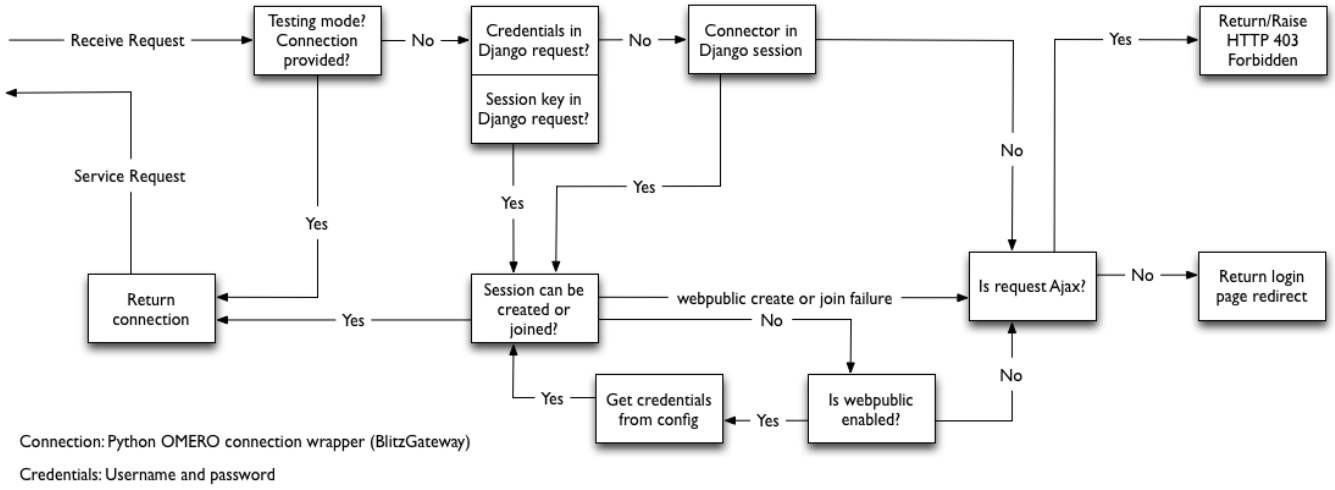


Figure 15.2: Logic flow for retrieving Blitz Gateway connection from HTTP request.

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 on the OMERO.web configuration page (see *Unix* or *Windows* version).

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.

15.7.2 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.core.urlresolvers import reverse
>>> reverse('url_name', kwargs={'object_id': 1})
```

15.7.3 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 found in the ..?...js code which should be included in all pages that require this functionality. Errors are handled as follows:

- 404 Simply display 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)
    else:
        ...   # render err_msg with a custom template
        return HttpResponseServerError(content)
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

15.8 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.

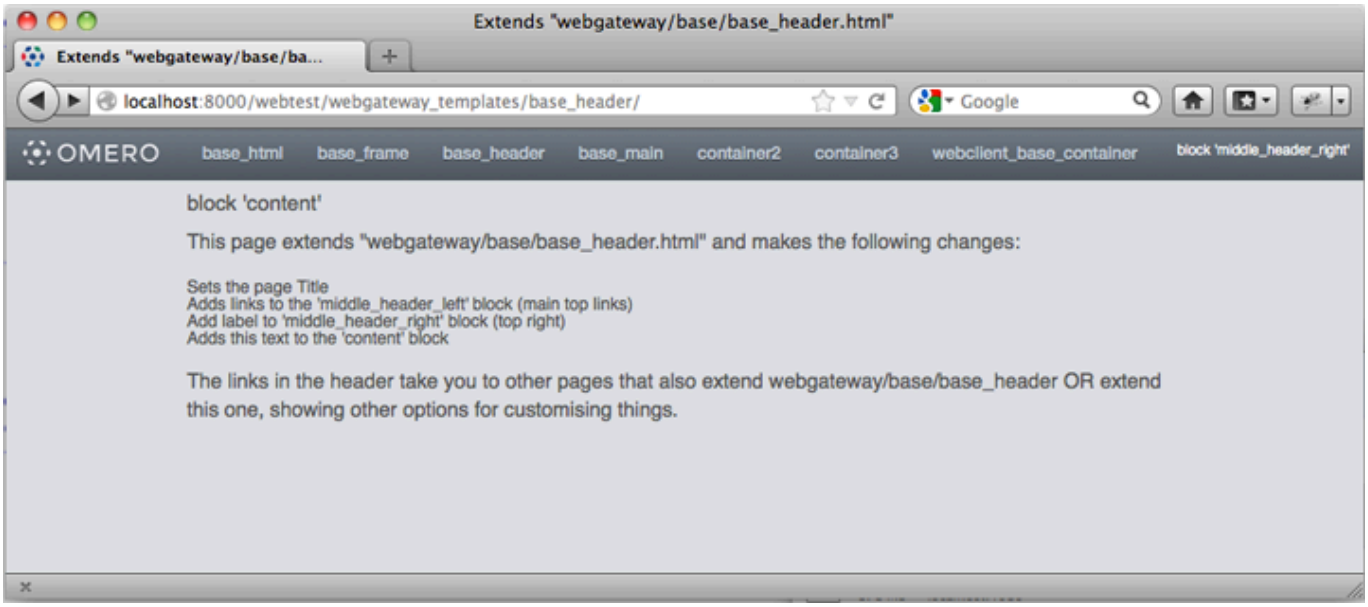


Figure 15.3: The `base_header.html` template extended in webtest with dummy content

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.

15.8.1 Django templates

We use Django templates for the OMERO.web pages. See docs here: https://docs.djangoproject.com/en/dev/ref/templates/ and template inheritance[15]. 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[16].

These templates are described in more detail below.

[15] https://docs.djangoproject.com/en/dev/topics/templates/#template-inheritance
[16] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/tools/OmeroWeb/omeroweb/webclient/templates/webclient/base

15.8.2 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.

15.8.3 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.

15.8.4 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:

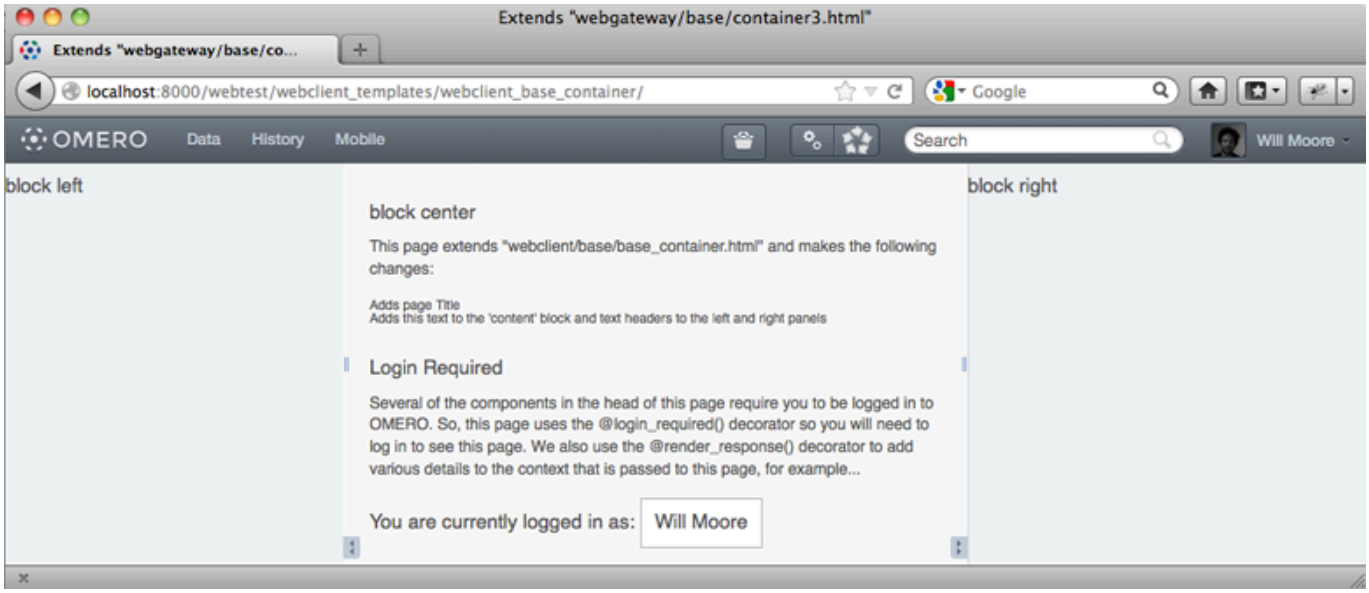


Figure 15.4: The `webclient/base/base_container.html` template extended in webtest with dummy content

- **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 webtest application. The location of these depends on your OMERO version:

- 4.4.0 - 4.4.4 - Look under omeroweb/webtest/templates/webtest/common. View live at <your-server-name>/webtest/common/base_header/
- After 4.4.4 - Look under omeroweb/webtest/templates/webtest/webgateway. View 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).

15.8.5 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[17] for full template details.

[17] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroWeb/omeroweb/webgateway/templates/webgateway/base/base_header.html

- 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[18] for full template details.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

15.9 Public data in OMERO.web

Since OMERO 4.4, the OMERO.web framework has supported 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 and data that is available to the defined 'public user'.

Here is how to go about setting this up on your OMERO.web install:

- Create a group with read-only permissions (name can be anything e.g. "public-data"). We use 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 (we will enter these below). Again, the First Name, Last Name, username and password can be anything you like.
- Enable the public user and set their username and password:

```
$ bin/omero config set omero.web.public.enabled True

$ bin/omero config set omero.web.public.user '<username>'

$ bin/omero config set omero.web.public.password '<password>'
```

- Set a URL filter for which the OMERO.web public user is allowed to navigate. Default: '^/(?!webadmin)' (Python regular expression). You probably do not want the whole webclient UI to be publicly visible (although you could do this).

  The idea is that you can create the public pages yourself since we do not provide them. For example, to allow only URLs that start with '/my_web_public' you would use:

[18] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroWeb/omeroweb/webgateway/templates/webgateway/base/container3.html

```
$ bin/omero config set omero.web.public.url_filter '/my_web_public'
```

To enable public access to view images in a public group in the webclient while still preventing data manipulation, use the following command:

```
$ bin/omero config set omero.web.public.url_filter '^/(?!webadmin|webclient/action/\w+|webclient/ann
```

If you simply want to enable the image viewer, making sure all data stays secure you would use:

```
$ bin/omero config set omero.web.public.url_filter '/webgateway'
```

Then you can access public images via the following link *http://your_host/webgateway/img_detail/IMAGE_ID/*. Please remember that public images must be in a public group where public user can access them.

Exotic matching techniques can be used but more explicit regular expressions are needed when attempting to filter based on a base URL:

```
'webtest' matches '/webtest' but also '/webclient/webtest'
'dataset' matches  '/webtest/dataset' and also '/webclient/dataset'
'/webtest' matches '/webtest…' but also '/webclient/webtest'
'/webtest' matches '/webtest…' but not '/webclient/webtest'
```

Set a server to connect to. Default: 1 (the first server in omero.web.server_list)

```
$ bin/omero config set omero.web.public.server_id 1
```

If you wish to mix public and restricted access to the system, the user can always access the login page using the following link *http://your_host/webclient/login/*.

CHAPTER

SIXTEEN

INSIGHT

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.1 Architecture

16.1.1 Logical view

OMERO.insight is logically organized in two layers

The **Agents** layer contains the logic to manage user interaction. It contains coarse grained components which we call **agents**, that are each responsible for a specific aspect of the application's functionality:

- The Data Manager provides the user with the GUI functionality to access their data, metadata and visualize large image sets.
- The Viewer is a tool to visualize and tune 5D images.
- The Measurement Tool is a tool to perform basic measurement.

Note: If you want to add a new agent, go to *How to build an agent*.

These agents are internally organized according to the MVC (Model-View-Controller[1]) pattern, PAC (Presentation-Abstraction-Control[2]) pattern, or a combination of the two. They rely on the services provided by the bottom layer, the **Container**, to accomplish their tasks.

The **Container** layer manages the agents life-cycle and provides them with services to:

- Communicate without having to know each other (*Event bus*).
- Access the OMERO Server (data management and image services).
- Transform entries in configuration files into objects and then access them (*Configuration*).
- Log messages (log service) and notify the user (user notification service) of runtime errors.
- Cache data (cache service).
- Provide a common top level window to plug their GUI's (*Taskbar*).

16.1.2 Initialization of Agents

Agents let the container create them and then manage their life-cycle. This is achieved through the use of a common interface, `Agent`, that all agents have to implement and by requiring every agent to have a public no-arguments constructor. The Agent interface plays the role of a Separated Interface (Fow[3]), decoupling the container from knowledge of concrete agents. This way, new agents can be plugged in.

[1] http://en.wikipedia.org/wiki/Model-view-controller
[2] http://en.wikipedia.org/wiki/Presentation-abstraction-control
[3] http://martinfowler.com/books

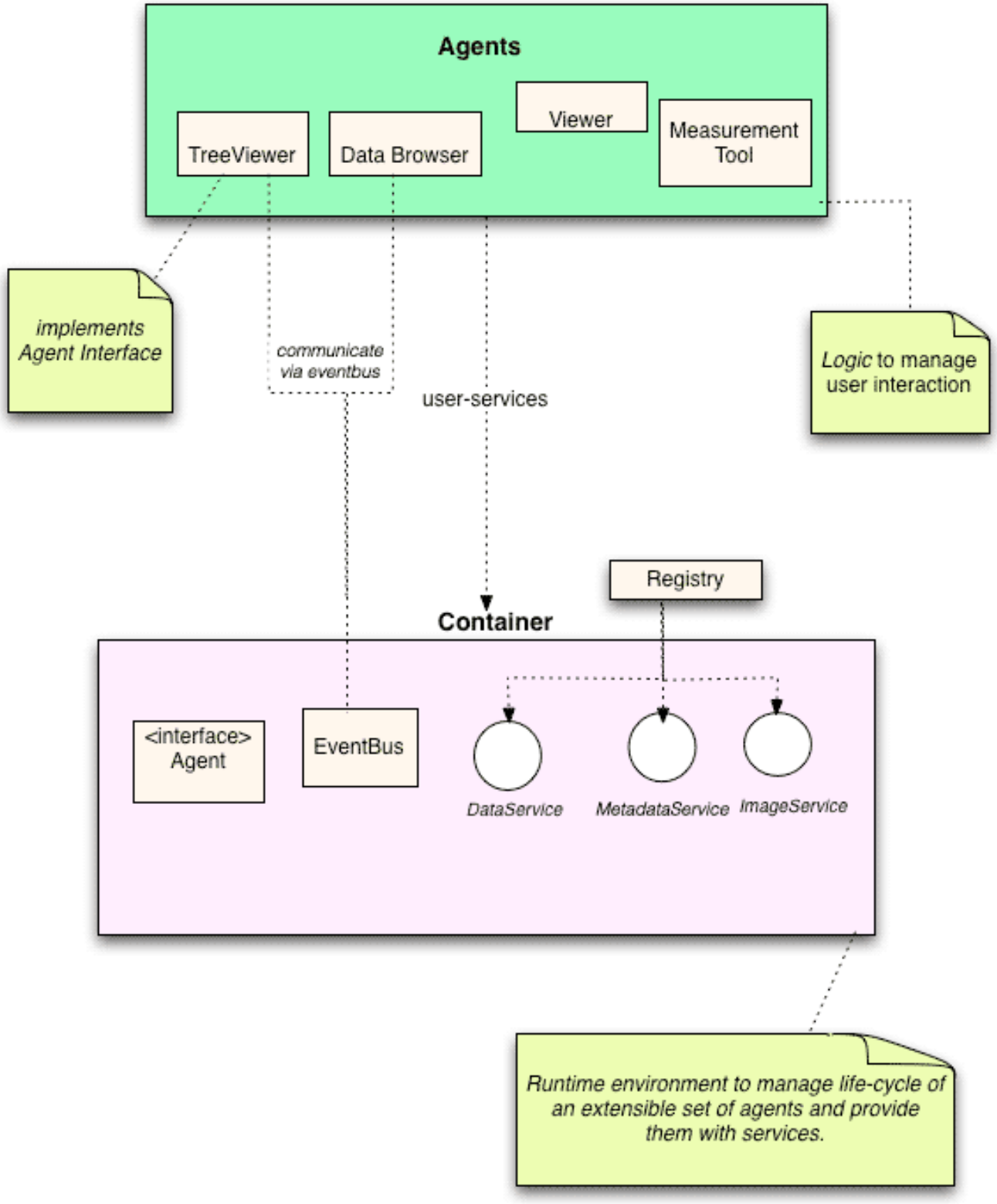


Figure 16.1: OMERO.insight agents and containers

At start-up the container finds out which are the agents' implementation classes from its configuration file, instantiates every agent by reflection (using the no-arguments constructor) and then reads each agent's configuration file (Fow[4]). The configuration entries in this file are turned into objects and collected into a map-like object, which is then passed to the agent. This map object also contains pointers to the container's services. We can think of this object as a Registry (Fow[5]).

[4] http://martinfowler.com/books
[5] http://martinfowler.com/books

There is one Registry containing pointers to the container's services for each agent, so configuration entries are private to each agent - container's services are shared among all agents though. Agents access the Registry object through the Registry interface.

The life-cycle of an agent is as follow:

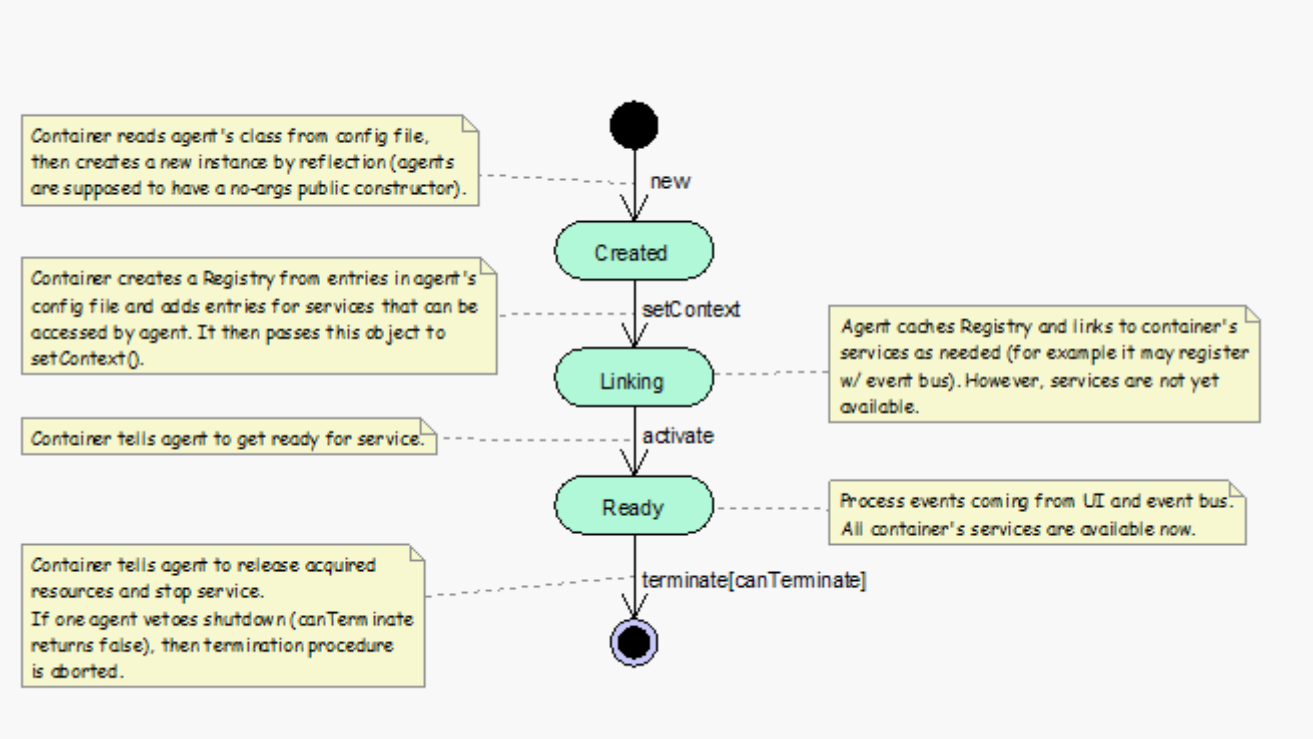


Figure 16.2: OMERO.insight agent lifecycle

16.1.3 Interaction among Agents

Interactions among agents are event-driven. Agents communicate by using a shared *Event bus* provided by the container. The event bus is an event propagation mechanism loosely based on the Publisher-Subscriber[6] pattern and can be regarded as a time-ordered event queue - if event A is posted on the bus before event B, then event A is also delivered before event B.

16.1.4 Process view

All agents run synchronously within the *Swing* dispatching thread. All container's services are called within Swing event handlers and thus run within the *Swing* dispatching thread. To see how to retrieve data from an OMERO server, go to the *Retrieve data from server* page.

See also:

Organization, *Event bus*

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.2 Configuration

The container provides a flexible and extensible configuration mechanism. Each agent has its own configuration file which is parsed at start-up by the container. The configuration entries in this file are turned into objects and collected into a map-like object, which is then passed to the agent. This map object also contains pointers to the container's services. Thus, we can think of this object as a *Registry*. There is one `Registry` for each agent, so configuration entries are private to each agent - container's services are shared among all agents though. The container also has its own configuration file and `Registry`.

[6] http://en.wikipedia.org/wiki/Publish/subscribe

The container maintains a set of predefined bindings that are used to convert a configuration entry into an object - such as a `String`, `Integer`, `Font`, `IconFactory`, etc. However, agents can specify custom handlers for converting a configuration entry into an object.

16.2.1 Structure

Configuration files are XML files which declares only two elements:

```
<element name="entry" minOccurs="0" maxOccurs="unbounded">
  <complexType>
    <simpleContent >
      <extension base="string">
        <attribute name="name" type="string" use="required"/>
        <attribute name="type"type="string"default="string"/>
        <simpleType>
          <restriction base="string">
            <enumeration value="string"/>
            <enumeration value="integer"/>
            <enumeration value="float"/>
            <enumeration value="double"/>
            <enumeration value="boolean"/>
          </restriction>
. . . close all tags
<element name="structuredEntry"minOccurs="0"maxOccurs="unbounded">
  <complexType>
    <sequence>
      <any maxOccurs="unbounded"/>
    </sequence>
    <attribute name="name" type="string" use="required"/>
    <attribute name="type" type="string" default="map"/>
  </complexType>
</element>
```

The *entry* and *structuredEntry* elements are used to specify name-value pairs in the actual configuration files. Both elements have a required name attribute whose content is used as a key for accessing the entry value from within the application. In the case of the entry element, the value is a string – this element can thus be used for "classic" name-value style, such as:

```
<entry name="ITEM">The item's value</entry>
```

On the other hand, the value of *structuredEntry* can be made up by any arbitrary sequence of tags.

In both cases (*entry* and *structuredEntry*), the entry value is returned to the application as an object and the type attribute dictates how the entry value is turned into an object. Only "string", "integer", "float", "double" and "boolean" may be used for the type attribute within the entry element, whose content is parsed into a Java `String`, `Integer`, `Float`, `Double` or `Boolean` object according to the content of the type attribute – if this attribute is missing, then "string" is assumed. For example, say you write the following into an agent's configuration file:

```
<entry name="/some/name" type="boolean">true</entry>
```

This will make a `Boolean` object (set to hold *true*) available to the agent – the key for accessing the object will be the string "/some/name".

Things work similarly for the *structuredEntry* element. In this case, the content of the type attribute can be specified to be the fully qualified name of the class that will handle the transformation of the entry value into an object. This is provided so that agents may specify custom handlers for custom configuration entries. For example, an agent's configuration file could contain the entry:

```
<structuredEntry name="/some/name" type="some.pkg.SomeHandler">
  <tag_1>aValue</tag_1>
```

```
  <tag_2>anotherValue</tag_2>
</structuredEntry>
```

In this case, an instance of `some.pkg.SomeHandler` will be created to transform the contents of the entry (that is *tag_1* and *tag_2*) into a custom object. Obviously enough, the tags contained within a *structuredEntry* have to be exactly the tags that the handler expects.

If no type attribute is specified, then it is assumed *type = "map"*, which results in the entry's contents being parsed into a `Map` object. Each child tag is assumed to be a simple tag with a string content, like in the following example:

```
<structuredEntry name="/some/name">
  <key_1>value_1</key_1>
  <key_2>value_2</key_2>
</structuredEntry>
```

Each child tag's name is a key in the map and the tag's content is its value – the above would generate the map: `(key_1, value_1), (key_2, value_2)`.

Some predefined structured entries are supplied by the container for common cases (icons and font entries) and for use by the container only (OMERO and agents entries). Here's an excerpt from the container's configuration file:

```
<container>
  <services>
    <structuredEntry name="/services/OMERODS" type="OMERODS">
      <port>1099</port>
    </structuredEntry>
  </services>
  <agents>
    <structuredEntry name="/agents" type="agents">
      <agent>
        <name>Viewer</name>
        <!-- The class tag specifies the FQN of the agent's class. -->
          <class>org.openmiscroscopy.shoola.agents.viewer.Viewer</class>
          <!-- The config tag specifies the name of the agent's configuration file.
               This file has to be placed in the config directory under the
               installation directory. -->
          <config>viewer.xml<config>
        </agent>
 . . . a similar entry for every other agent
    </structuredEntry>
  </agents>
  <resources>
    <iconFactories>
      <!-- This entry can be used in agents' configuration files as well.
       It is turned into an instance of:
       org.openmicroscopy.shoola.env.config.IconFactory
       This object can then be used to retrieve any image file within
       the directory pointed by the location tag. -->
      <structuredEntry name="/resources/icons/DefaultFactory" type="icons">
       <!-- The location tag specifies the FQN of the package that contains the icon files. -->
        <location>org.openmicroscopy.shoola.env.ui.graphx</location>
    </structuredEntry>
 . . . more similar entries
    </iconFactories>
    <fonts>
      <!-- This entry can be used in agents' configuration files as well.
           It is turned into an instance of java.awt.Font. -->
      <structuredEntry name="/resources/fonts/Titles" type="font">
        <family>SansSerif</family>
        <size>12</size>
        <style>bold</style>
```

```
      </structuredEntry>
  . . . more similar entries
    </fonts>
    </resources>
</container>
```

The configuration parser only takes the *entry* and *structuredEntry* tags into account and ignores all the others. It may be useful to group sets of related entries together, as shown above.

The classes that encompass the machinery for parsing configuration files and building registries are depicted by the following UML class diagram.

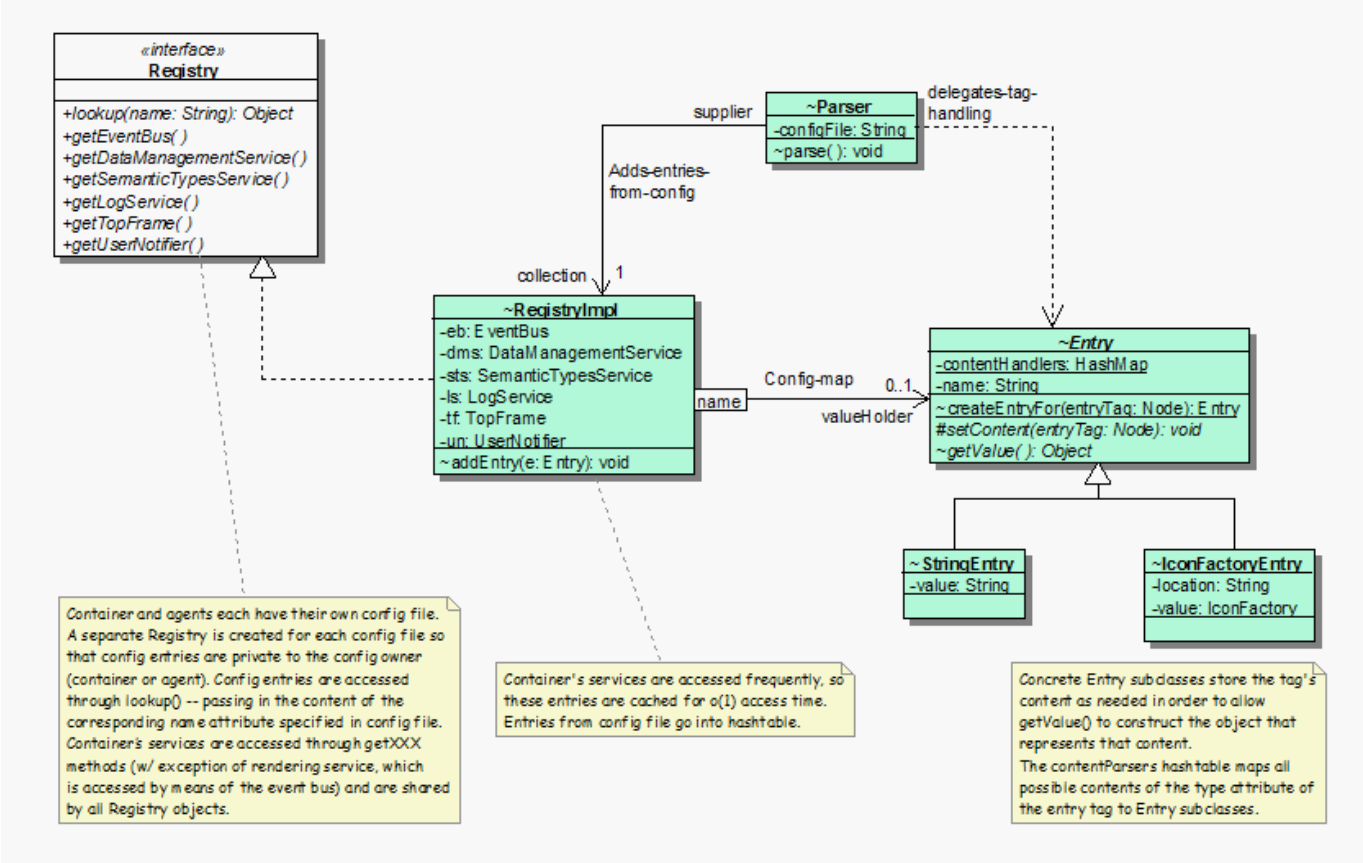


Figure 16.3: OMERO.insight configuration

The `Entry` abstract class sits at the base of a hierarchy of classes that represent entries in configuration files. It represents a name-value pair, where the name is the content of the *name* attribute of a configuration entry (which is stored by the `name` field) and the value is the object representing the entry's content. As the logic for building an object from the entry's content depends on what is specified by the *type* attribute, this class declares an abstract `getValue` method which subclasses implement to return the desired object – we use polymorphism to avoid conditional logic. So we have subclasses (`StringEntry`, `IntegerEntry`, `IconFactoryEntry`, etc.) to handle the content of an entry tag (either *entry* or *structuredEntry*) in correspondence of each predefined value of the type attribute ("string", "integer", "icons", and so on). Given an entry tag, the `createEntryFor` static method (which can be considered a Factory Method) creates a concrete `Entry` object to handle the conversion of that tag's content into an object. Subclasses of `Entry` implement the `setContent` method to grab the tag's content, which is then used for building the object returned by the implementation of `getValue()`.

The `Registry` Interface declares the operations to be used to access configuration entries and container's services.

The `RegistryImpl` class implements the `Registry` interface. It maintains a map of `Entry` objects, which are keyed by their name attribute and represent entries in the configuration file. It also maintains references to the container's services into member fields – as services are accessed frequently, this ensures *o(1)* access time.

The `Parser` class is in charge of parsing a configuration file, extracting entries (only *entry* and *structuredEntry* tags are taken into account), obtain an `Entry` object to represent each of those entries and add these objects to a given `RegistryImpl` object.

16.2.2 Dynamics

How a configuration file is parsed and the corresponding Registry is built:

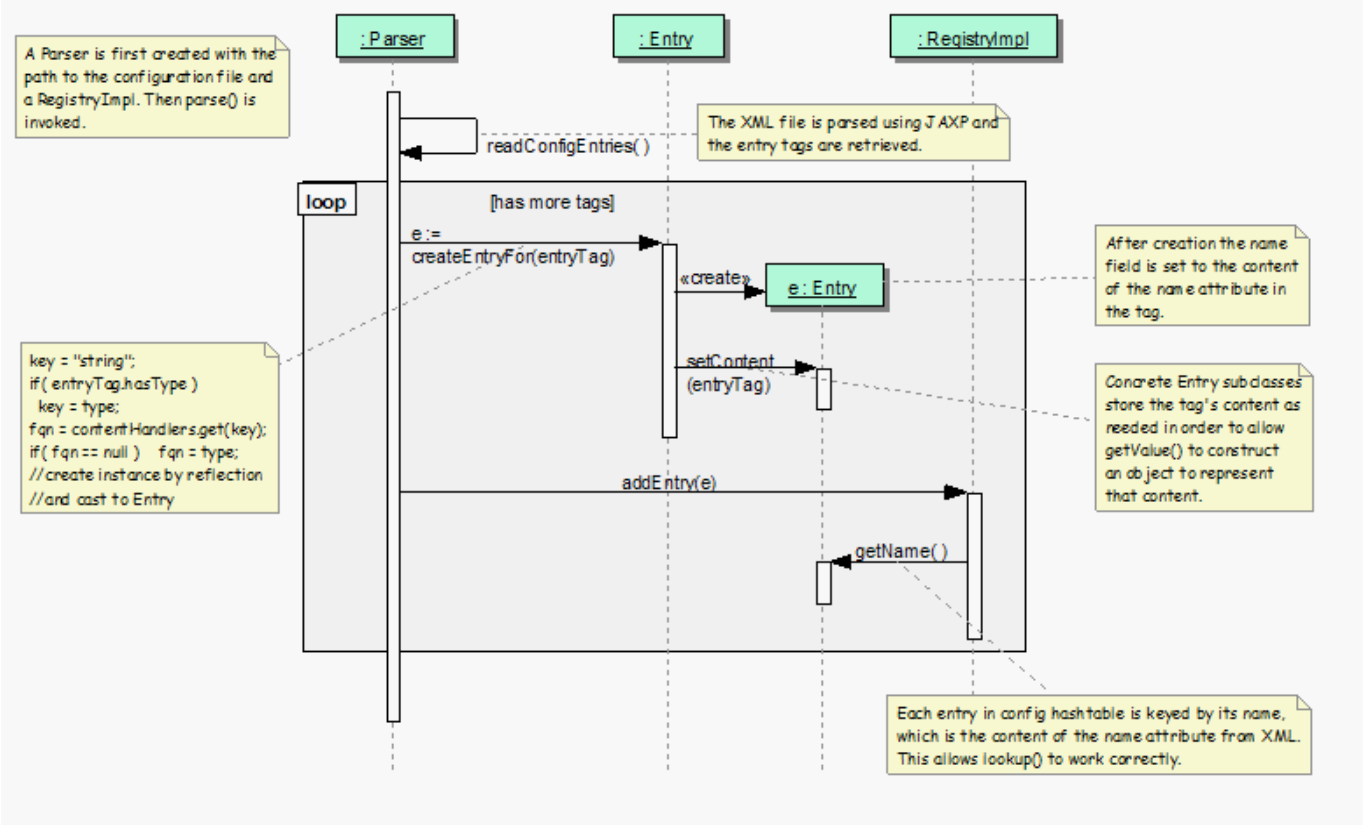


Figure 16.4: Parsing configuration files

A `RegistryImpl` object is created with an empty map. Then a `Parser` object is created passing the path to the configuration file and the `RegistryImpl` object. At this point `parse()` is invoked on the `Parser` object. The configuration file is read (the XML parsing is handled by built-in JAXP libraries) and, for each configuration entry (that is, either *entry* or *struturedEntry* tag), `createEntryFor()` is called to obtain a concrete `Entry` object, which will handle the conversion of the tag's content into an object. This `Entry` object is then added to the map kept by the `RegistryImpl` object.

In order to find out which class is in charge of handling a given tag, the `Entry` class maintains a map, `contentHandlers`, whose keys are the predefined values used for the type attribute ("string", "integer", "icons", etc.) and values are the fully qualified names of the handler classes. Given a tag, `createEntryFor()` uses the content of the type attribute (or "string" if this attribute is missing) to look up the class name in the map and then creates an instance by reflection - all `Entry`'s subclasses are supposed to have a no-args constructor. If the class name is not found in the map, then the content of the type attribute is assumed to be a valid fully qualified name of an `Entry`'s subclass. This allows for agents to specify custom handlers – as long as the handler extends `Entry` and has a public no-args constructor.

Notice that the `RegistryImpl` object adds the couple `(e.getName(), e)` to its map. Because the `Entry` class takes care of setting the name field to the content of the name attribute within the entry tag, the application code can subsequently access e by specifying the value of the name attribute to `lookup()`. The above outlined process is repeated for each configuration file so that the configuration entries of each agent (and the container) are kept in separate objects – a `RegistryImpl` is created every time. Because every agent is then provided with its own `RegistryImpl` object, the configuration entries are private to each agent. However, the container configures all `RegistryImpl` objects with the same references to its services.

See also:

Directory contents

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.3 Contributing to OMERO.insight

16.3.1 Getting started with OMERO.insight

Getting started with OMERO.insight entails that you have an *OMERO.server* already deployed.

16.3.2 Installing from source

Since January 2011, the OMERO.insight code base is part of the OMERO code base. See *Installing OMERO from source*, to check out code using http://git.openmicroscopy.org.

Requirements

- Install a Java 6 or Java 7 Development Kit (JDK), available from Java SE Downloads[7] and required for both the OMERO server and client code. Set the `JAVA_HOME` environment variable to your JDK installation.

Running code

It is helpful to set up the project in Eclipse[8]. Because the OMERO Java and Python source files are encoded in UTF-8, ensure that the encoding in Eclipse (*Preferences* → *General* → *Workspace* → *Text file encoding*) is also set to UTF-8.

Build system

Ant

The compilation, testing, launch, and delivery of the application are automated by means of an Ant[9] build file, located under the `build` directory (See *Directory contents*). Move to the build directory and, from the command line, enter:

```
java build
```

This will display the available targets to compile, run, test, and create a distribution bundle. Use the target you wish, for example:

```
java build all
```

Because all the tools needed to build the software are already included in the build directory, you do not need to have Ant on your machine. If you wish to use Ant instead, you can still do it by using the `build.xml` file under the `build` directory. However, there are some dependencies to satisfy before; these are clearly documented in the `build.xml` file itself.

Jenkins

The OME project currently uses Jenkins[10] (formerly known as hudson) as a continuous integration server available here[11]. OMERO.insight is built as part of the "OMERO" job[12].

Jenkins checks for SVN changes every 15 minutes and executes:

[7] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[8] http://www.eclipse.org
[9] http://ant.apache.org/
[10] http://jenkins-ci.org
[11] http://ci.openmicroscopy.org/
[12] http://ci.openmicroscopy.org/job/OMERO-trunk/

```
export JBOSS_HOME=$HOME/root/opt/jboss
export JAVA_OPTS=&amp;quot;-Xmx600M -Djavac.maxmem=600M -Djavadoc.maxmem=600M -XX:MaxPermSize=256m&amp;quot;

#
# Build
#
J=7 java $JAVA_OPTS omero build-all
# integration unfinished


#
# Documentation and build reports
#
java $JAVA_OPTS omero -f components/antlib/resources/release.xml -Dbasedir=. javadoc
java $JAVA_OPTS omero findbugs # separate call to prevent PermGen OOM
java $JAVA_OPTS omero coverage


#
# Prepare a distribution
#
rm -f OMERO.insight-build*.zip
java -Domero.version=build$BUILD_NUMBER omero zip
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.4 Directory contents

The repository of the software artifacts is organized as follows:

- `build`: This directory contains the tools to compile, run, test, and deliver the application.
- `config`: Various configuration files required by the application to run.
- `docgen`: Documentation artifacts that are used to build actual documents. These are organized in two sub-directories: javadoc and xdocs. The former only contains resources (like CSS files) to generate programmer's documentation – the actual documentation contents are obtained from the source code. The latter contains both resources (like stylesheets and DHTML code) to generate all other kinds of documentation (like design and users documents) and the actual documentation contents in the form of XML/HTML files.
- `launch`: Launcher scripts and installation instructions bundled with the default application distribution file. Its sub-dirs contain further resources to build platform-specific distributions.
- `SRC`: Contains the application source files.
- `TEST`: The test code.
- `README`: The README file.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.5 Event bus

Interactions among agents are event-driven. Agents communicate by using a shared event bus provided by the container. The event bus is an event propagation mechanism loosely based on the Publisher-Subscriber[13] pattern and can be regarded as a time-ordered event queue - if event A is posted on the bus before event B, then event A is also delivered before event B.

Events are fired by creating an instance of a subclass of *AgentEvent* and by posting it on the event bus. Agents interested in receiving notification of *AgentEvent* occurrences implement the *AgentEventListener* interface and register with the event bus.

[13] http://en.wikipedia.org/wiki/Publish/subscribe

This interface has a callback method, `eventFired`, that the event bus invokes in order to dispatch an event. A listener typically registers interest only for some given events - by specifying a list of *AgentEvent* subclasses when registering with the event bus. The event bus will then take care of event de-multiplexing - an event is eventually dispatched to a listener only if the listener registered for that kind of event.

16.5.1 Structure

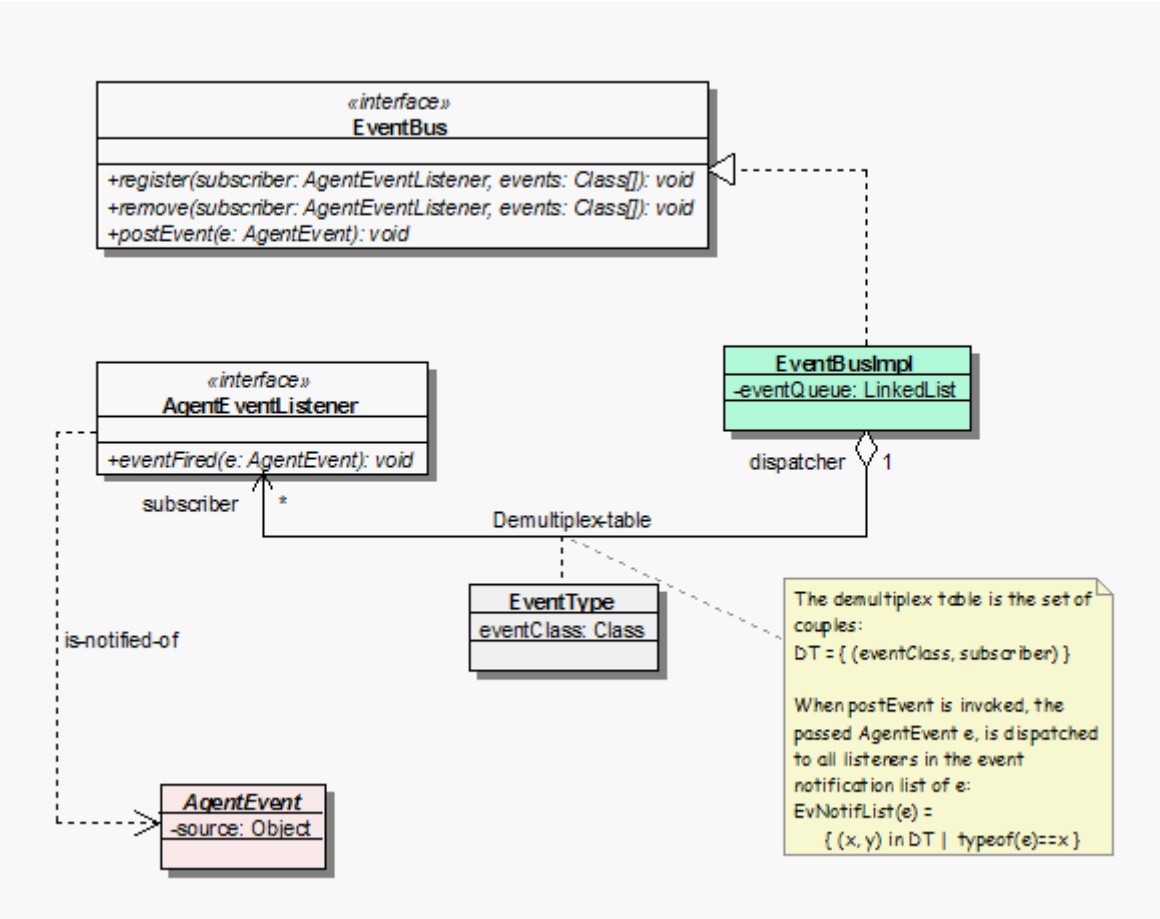


Figure 16.5: OMERO.insight event bus

EventBus

- Defines how client classes access the event bus service.
- A client object (subscriber) makes/cancels a subscription by calling `register()`/`remove()`.
- A client object (publisher) fires an event by calling `postEvent()`.

AgentEvent

- Ancestor of all classes that represent events.
- Source field is meant to be a reference to the publisher that fired the event.
- An event is "published" by adding its class to the events package within the agents package.

AgentEventListener

- Represents a subscriber to the event bus.
- Defines a callback method, `eventFired`, that the event bus invokes in order to dispatch an event.

EventBusListener

- Concrete implementation of the event bus.
- Maintains a de-multiplex table to keep track of what events have to be dispatched to which subscribers.

16.5.2 In action

- When a subscriber invokes the register or remove method, the de-multiplex table is updated accordingly and then the event bus returns to idle.
- When a publisher invokes `postEvent()`, the event bus enters into its dispatching loop and delivers the event to all subscribers in the event notification list.
- Time-ordered event queue - if event A is posted on the bus before event B, then event A is also delivered before event B.
- Dispatching loop runs within same thread that runs the agents (*Swing* dispatching thread).

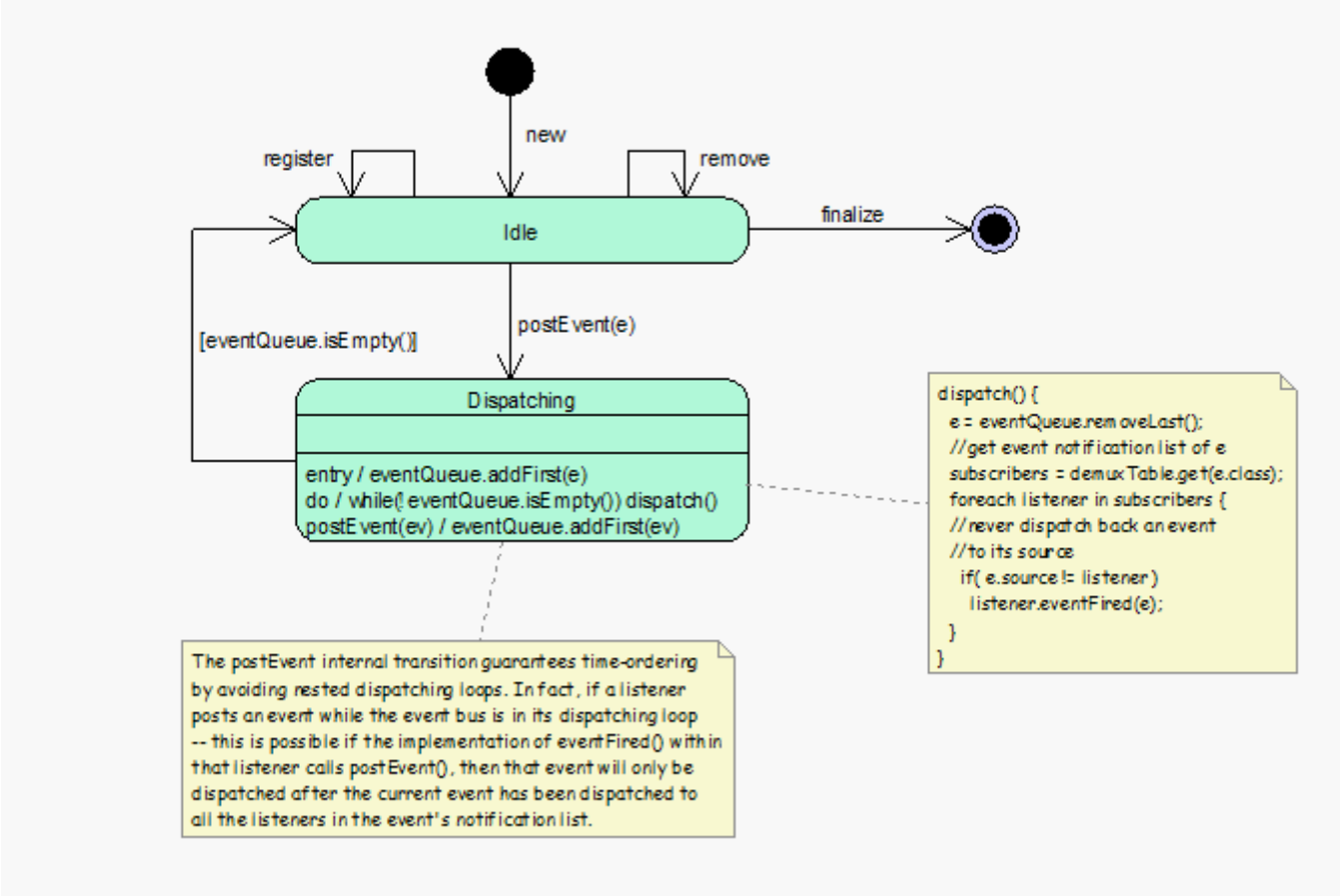


Figure 16.6: OMERO.insight event dispatching

16.6 Event

16.6.1 Structure

We devise two common categories of events:

- Events that serve as a notification of state change. Usually events posted by agent to notify other agents of a change in its internal state.
- Events that represent invocation requests and completion of asynchronous operations between agents and some services.

In the first category fall those events that an agent posts on the event bus to notify other agents of a change in its internal state. Events in the second category are meant to support asynchronous communication between agents and internal engine. The *AgentEvent* class, which represents the generic event type, is sub-classed in order to create a hierarchy that represents the above categories. Thus, on one hand we have an abstract *StateChangeEvent* class from which agents derive concrete classes to represent state change notifications. On the other hand, the *RequestEvent* and *ResponseEvent* abstract classes are sub-classed by the container in order to define, respectively, how to request the asynchronous execution of an operation and how to represent its completion. We use the

Asynchronous Completion Token pattern to dispatch processing actions in response to the completion of asynchronous operations.

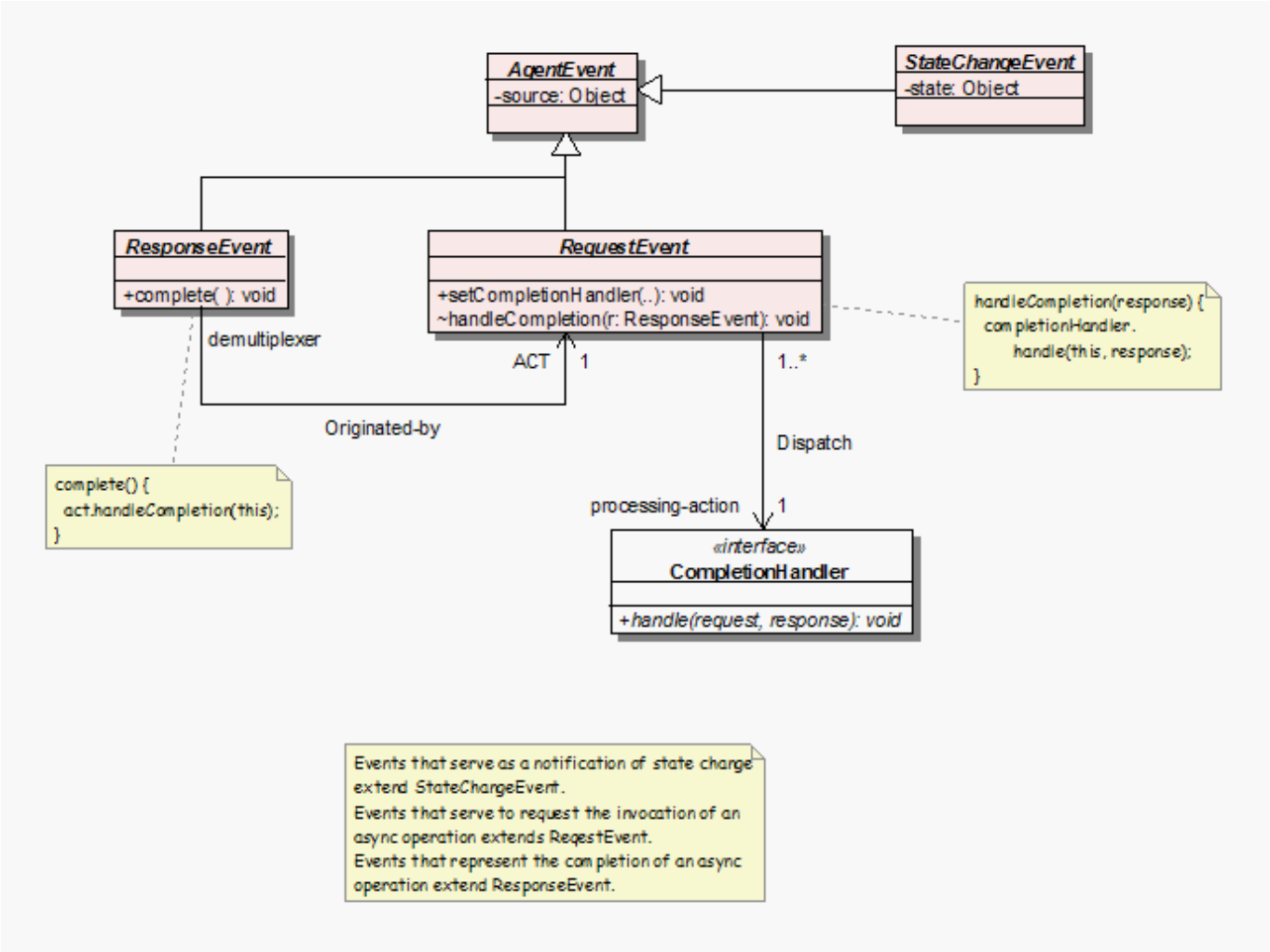


Figure 16.7: OMERO.insight events

StateChangeEvent

- Ancestor of all classes that represent state change notifications.
- Its state field can be used to carry all state-change information.

RequestEvent

- Abstractly represents a request to execute an asynchronous operation.
- A concrete subclass encapsulates the actual request.
- Knows how and which processing action to dispatch upon completion of the asynchronous operation.

CompletionHandler

- Represents a processing action.
- Allows for all processing action to be treated uniformly.

ResponseEvent

- Abstractly represents the completion of an asynchronous operation.
- A concrete subclass encapsulates the result of the operation, if any.
- Knows the *RequestEvent* object that originated it.
- Knows how to activate the de-multiplexing of a completion event to the processing action.

16.6.2 In action

Follow a concrete example:

```
//Somewhere in the Data Manager
//Request to View an image

EventRequest req = new ViewImage((ImageData) image, null)
//Request the execution of the view call.
eventBus.postEvent(req);


//Somewhere in the Viewer Agent
public void eventFired(AgentEvent e)
{
    if (e instanceof ViewImage) handleViewImage((ViewImage) e);
}
```

A concrete *RequestEvent* encapsulates a request to execute an asynchronous operation. Asynchrony involves a separation in space and time between invocation and processing of the result of an operation: we request the execution of the operation at some point in time within a given call stack (say in `methodX` we make a new request and we post it on the event bus). Then, at a later point in time and within another call stack (`eventFired` method), we receive a notification that the execution has completed and we have to handle this completion event - which mainly boils down to doing something with the result, if any, of the operation. Recall that the *ResponseEvent* class is used for representing a completion event and a concrete subclass carries the result of the operation, if any. After the operation has completed, a concrete *ResponseEvent* is put on the event bus so that the object which initially made the request (often an agent, but, in this context, we will refer to it as the initiator, which is obviously required to implement the *AgentEventListener* interface and register with the event bus) can be notified that execution has completed and possibly handle the result. Thus, at some point in time the initiator's eventFired method is called passing in the response object.

Now the initiator has to find out which processing action has to be dispatched to handle the response. Moreover, the processing action often needs to know about the original invocation context - unfortunately, we cannot relinquish the original call stack (`methodX` is gone). The solution is to require that a response be linked to the original request and that the initiator link a request to a completion handler (which encapsulates the processing action) before posting it on the event bus (this explains the fancy arrangement of the *RequestEvent*, *ResponseEvent* and *CompletionHandler*).

This way de-multiplexing matters are made very easy for the initiator. Upon reception of a completion event notification, all what the initiator has to do is to ask the response object to start the de-multiplexing process - by calling the complete method. This method calls `handleCompletion()` on the original request, passing in the response object. In turn, `handleCompletion()` calls the handle method on its completion handler, passing in both the request and the response. The right processing action has been dispatched to handle the response. Also, notice that the completion handler is linked to the request in the original invocation context, which makes it possible to provide the handler with all the needed information from the invocation context. Moreover, both the original request and the corresponding response are made available to the completion handler. This is enough to provide the completion handler with a suitable execution context - all the needed information from the original call stack is now available to the processing action.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.7 How to build an agent

An agent is created and managed by the container. Building an agent is done in two steps:

- write the agent code
- declare the agent in the `container.xml` file located in the `config` directory

The agent intercepts the events posted on the event bus.

Note: When a new version of the software is delivered, make sure you keep the `container.xml` shipped with the application and add the new agent entry to it.

16.7.1 Writing code

The following example creates a concrete agent `MyBrowserAgent`:

- Create a `myBrowser` package in the `agents` package.
- Create a class `MyBrowserAgent`, this class **MUST** implement the `Agent` interface to be initialized and the `AgentListener` to interact with other agents.

```
public class MyBrowserAgent
    implements Agent, AgentEventListener
{

    /** Reference to the registry. */
    private static Registry         registry;

    //no-arguments constructor required for initialization
    public MyBrowserAgent() {}

    //Follow methods required by the Agent Interface

    //No-op implementation in general
    public void activate()
    {
        //this method will be invoked during the activation by the container
    }

    //invoked before shutting down the application
    public boolean canTerminate() { return true; }

    //not yet implemented: invoked when shutting down the application
    public Map<String, Set> hasDataToSave() { return null; }

    //invoked while shutting down the application
    public void terminate() {}

    public void setContext(Registry ctx)
    {
        //Must be a reference to the Agent Registry to access services.
        registry = ctx;

        //register the events the agent listens to e.g. BrowseImage
        EventBus bus = registry.getEventBus();
        bus.register(this, BrowseImage.class);

    }



    //Follow methods required by the AgentEventListener Interface
    public void eventFired(AgentEvent e)
    {
        if (e instanceof BrowseImage) {
            //Do something
            browseImage((BrowseImage) e);
        }
    }

}
```

Where to create the `BrowseImage` event

- Create a `myBrowser` package in the `agents.events` package.

- Create a `BrowseImage` event in the `myBrowser` package.

```
public class BrowseImage
    extends RequestEvent
{

    /** The id of the image to browse. */
    private long imageID;

    /**
     * Creates a new instance.
     *
     * @param imageID The id of image to view.
     */
    public BrowseImage(long imageID)
    {
        if (imageID < 0)
            throw new IllegalArgumentException("ImageID not valid.");
        this.imageID = imageID;
    }


    /**
     * Returns the ID of the image to browse.
     *
     * @return See above.
     */
    public long getImageID() { return imageID; }

}
```

Listening to the `BrowseImage` event

To listen to events posted on the event bus, the agent **MUST** implement the `AgentListener` Interface and register the events to listen to.

- Register `BrowseImage` in the `setContext(Registry)` method of the `Agent` interface.
- Listen to `BrowseImage` in the `eventFired(AgentEvent)` method of the `AgentListener` interface.

For example, when clicking on an image in the Data Manager, the following event is posted:

```
EventBus bus = registry.getEventBus();
bus.post(new BrowseImage(imageID));
```

The `MyBrowserAgent` handles the event

```
public void eventFired(AgentEvent e)
{
    if (e instanceof BrowseImage) {
        //Do something
        browseImage((BrowseImage) e);
    }
}
```

Creating an agent's view

See *How to build an agent's view*

16.7.2 Declaring the agent

The `MyBrowserAgent` needs to be declared in the `container.xml`.

- Open the `container.xml` located in the `config` folder (see *Directory contents*).
- Add the following:

```
<agents>
    <structuredEntry name="/agents" type="agents">

      <!-- NOTE FOR DEVELOPERS
           Add an agent tag for each of your Agents.
           The name tag specifies the human-readable name of the Agent.
           The active tag specifies if the agent is turned on or off.
           Set to true to turn the agent on, false otherwise.
           The class tag specifies the FQN of the Agent class.
           The config tag specifies the name of the Agent's
           configuration file within the config directory.
       -->
      <agent>
         <name>My Browser</name>
         <active>true</active>
         <class>org.openmicroscopy.shoola.agents.mybrowser.MyBrowserAgent</class>
         <config>mybrowser.xml</config>
       </agent>
...
    </structuredEntry>
  </agents>
```

- Create a `mybrowser.xml` and add it to the `config` directory:

```
<?xml version="1.0" encoding="utf-8"?>
<agent name="My Browser">
   <resources>
     <iconFactories>
    <!-- This entry is turned into an instance of:
       org.openmicroscopy.shoola.env.config.IconFactory
       This object can then be used to retrieve any image file within
       the directory pointed by the location tag. -->
       <structuredEntry name="/resources/icons/Factory" type = "icons">
         <!-- The location tag specifies the FQN of the package that contains
         the icon files. -->
         <location>org.openmicroscopy.shoola.agents.myBrowser.graphx</location>
       </structuredEntry>

     </iconFactories>
     <fonts>
       <!-- This entry is turned into an instance of java.awt.Font. -->
       <structuredEntry name="/resources/fonts/Titles" type="font">
         <family>SansSerif</family>
         <size>12</size>
         <style>bold</style>
       </structuredEntry>
     </fonts>
</resources>
</agent>
```

The file `mybrowser.xml` allows the agent to define specific parameters.

See also:

Organization, *Retrieve data from server*

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.8 How to build an agent's view

This section explains how a view of the agent is created. All our agents follow the same approach. To see the code while reading the notes, go to components/insight/SRC/org/openmicroscopy/shoola/agents/treeviewer/view[14].

Using the previous example `MyBrowserAgent` (see *How to build an agent*):

1. Create a `view` package in the `mybrowser` package.
2. Create the following classes `MyBrowser` (interface), `MyBrowserComponent`, `MyBrowserModel`, `MyBrowserControl`, and `MyBrowserUI`. If you browse the source code, you will notice that we usually have a class used as a toolbar and a class used as a status bar. Both classes are initialized by the `BrowserUI`. For clarity, they have been omitted in the following diagram.
3. Create a `MyBrowserFactory`. This class keeps track of the `MyBrowser` instances created and not yet discarded. A component is only created if none of the tracked ones is displaying the data, otherwise the existing component is recycled.

16.8.1 Typical life-cycle of an agent view

The object is first created using the `MyBrowserFactory`

```
//Somewhere in the MyBrowserFactory code

    /** The sole instance. */
    private static final MyBrowserFactory  singleton = new MyBrowserFactory();

    /**
     * Returns a viewer to display the specified images.
     *
     * @param images The <code>ImageData</code> objects.
     */
    public static MyBrowser getViewer(Set<ImageData> images)
    {
        MyBrowserModel model = new MyBrowserModel(images);
        return singleton.getViewer(model);
    }


    /**
     * Creates or recycles a viewer component for the specified
     * <code>model</code>.
     *
     * @param model The component's Model.
     * @return A {@link MyBrowser} for the specified <code>model</code>.
     */
    private MyBrowser getViewer(MyBrowserModel model)
    {
        Iterator v = viewers.iterator();
        MyBrowserComponent comp;
        while (v.hasNext()) {
            comp = (MyBrowserComponent) v.next();
            if (model.isSameDisplay(comp.getModel())) {
                comp.refresh(); //refresh the view.
                return comp;
            }
        }
        comp = new MyBrowserComponent(model);
```

[14] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/insight/SRC/org/openmicroscopy/shoola/agents/treeviewer/view

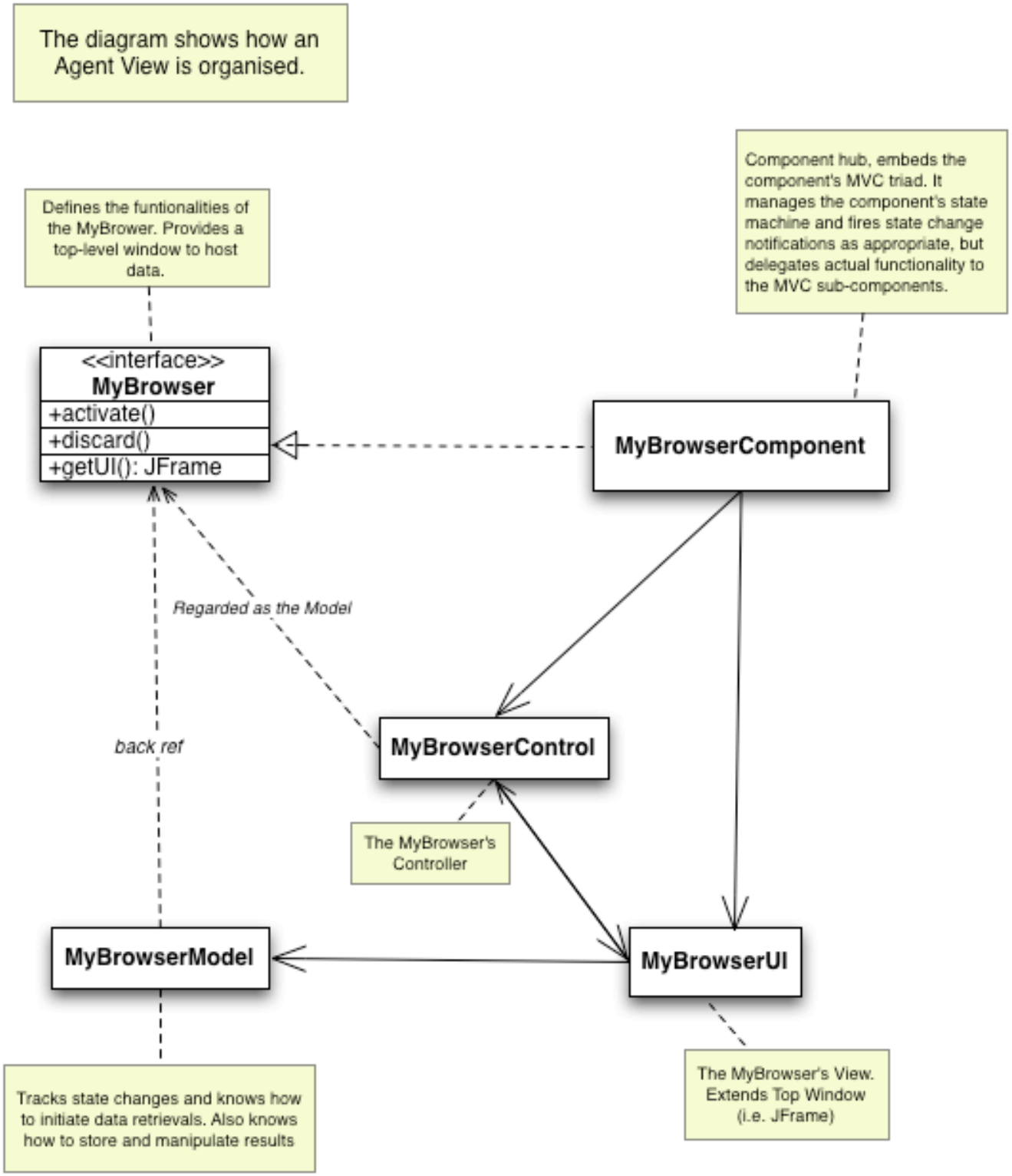


Figure 16.8: OMERO.insight agent view

```
        comp.initialize();
        comp.addChangeListener(this);
        viewers.add(comp);
        return comp;
    }
```

After creation, the object is in the `MyBrowser#NEW` state and is waiting for the `MyBrowser#activate()` method to be called. Such a call usually triggers loading of the objects on the server. The object is now in the `MyBrowser#LOADING` state. After all the data have been retrieved, the object is in the `MyBrowser#READY` state and the data display is built and set on screen.

When the user quits the window, the `MyBrower#discard()` method is invoked and the object transitions to the `MyBrowser#DISCARDED` state. At which point, all clients should de-reference the component to allow for garbage collection.

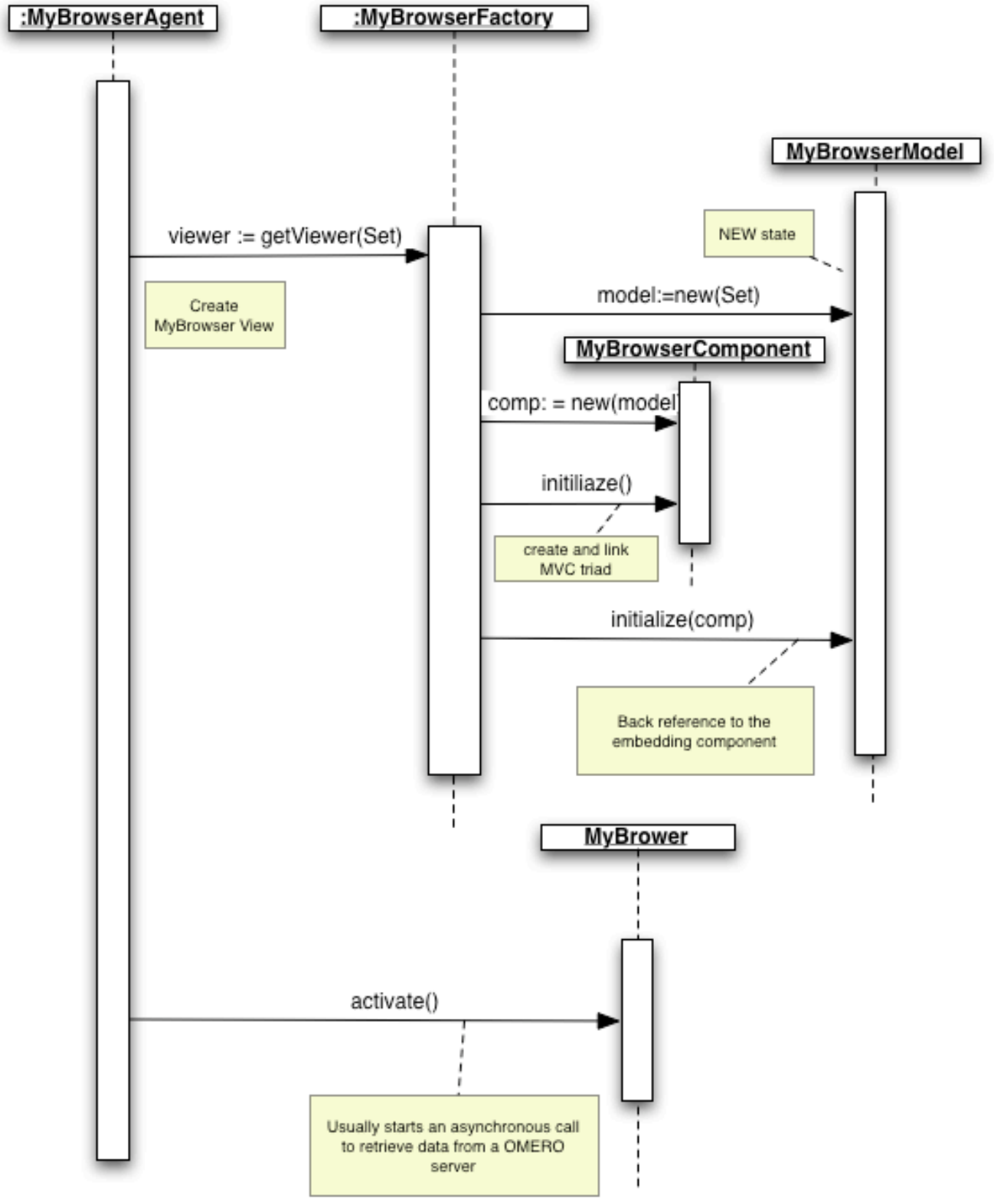


Figure 16.9: OMERO.insight agent view initialization

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.9 Retrieve data from server

To retrieve data stored in an OMERO server, Agents can either:

- directly access a Data service (container service) through their `Registry` (in which case, the call happens in the *Swing* dispatching thread, so it is not possible to give user feedback by showing a progress bar for example):

```
OmeroDataService service = registry.getDataService();
service.getServerName()
```

- or retrieve data asynchronously using a **Data Services View**

16.9.1 Data services view

Usage

A data services view is a logical grouping of data and operations that serve a specific purpose, for example to support dataset browsing by providing easy access to datasets, thumbnails, tags, etc. A data services view is defined by an interface that extends `DataServiceView` and consists of a collection of asynchronous calls that operate on (possibly) large portions of a data model in the background.

Agents obtain a reference to a given view through their registry by specifying the view's defining interface as follows (note the *required* cast on the returned reference):

```
XxxView view = (XxxView) registry.getDataServicesView(XxxView.class);
```

`XxxView` is obviously a made up name for one of the sub-interfaces of `DataServiceView` contained in this package. All calls are carried out asynchronously with respect to the caller's thread and return a `CallHandle` object which can be used to cancel execution. This object is then typically linked to a button so that the user can cancel the task, like in the following example:

```
final CallHandle handle = view.loadSomeDataInTheBg(observer);

   //The above call returns immediately, so we don't have to wait.
   //While the task is carried out, we allow the user to change
   //her mind and cancel the task:

   cancelButton.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent e) {
           handle.cancel();
       }
   });
```

The `observer` argument to the above call is an instance of `AgentEventListener` (in `env.event`). Normally all calls within a view allow to specify this argument, which is used to provide the caller with feedback on the progress of the task and with its eventual outcome.

Specifically, as the computation proceeds in the background, `DSCallFeedbackEvents` (in `env.data.events`) are delivered to the `observer`. These event objects have a status field which contains a textual description of the activity currently being carried out within the computation and a progress indicator which is set to the percentage of the work done so far. So the indicator will be `0` for the first feedback event and, if the computation runs to completion, `100` for the last feedback event, which will always have its status field set to `null` – note that a `null` status is also possible for the previous events if no description was available at the time the event was fired. Moreover, any partial result that the computation makes available will be packed into the feedback event.

It is important to keep in mind that the computation may not run to completion – either because of an exception within the computation or because the agent cancels execution – `CallHandle.cancel()` (in `env.data.views`). In both cases, the feedback notification will not run to completion either. However, in any case a final `DSCallOutcomeEvent` (in `env.data.events`) is delivered to the `observer` to notify of the computation outcome – the event's methods can be used to find out the actual outcome and retrieve any result or exception. Every call documents what is the returned object and what are the possible exceptions so that the caller can later cast the returned value or exception as appropriate.

Here is the code for a prototypical `observer`:

```
public void eventFired(AgentEvent ae)
{
    if (AE instanceof DSCallFeedbackEvent) {  //Progress notification.
        update((DSCallFeedbackEvent) AE);  //Inform the user.
    } else {  //Outcome notification.
        DSCallOutcomeEvent oe = (DSCallOutcomeEvent) AE;
        switch (oe.getState()) {
        case DSCallOutcomeEvent.CANCELLED:  //The user cancelled.
            handleCancellation();
            break;
        case DSCallOutcomeEvent.ERROR:  //The call threw an exception.
            handleException(oe.getException());
            break;
        case DSCallOutcomeEvent.NO_RESULT:  //The call returned no value.
            handleNullResult();
            break;
        case DSCallOutcomeEvent.HAS_RESULT:  //The call returned a value.
            handleResult(oe.getResult());
        }
    }
}
```

Because the logic is likely to be common to most of the observers, the `DSCallAdapter` (in `env.data.events`) class factors it out to provide a more convenient way to write observers. Back to our previous example, the observer could look something like the following:

```
observer = new DSCallAdapter() {
     public void update(DSCallFeedbackEvent fe) {  //Received some feedback.
         String status = fe.getStatus();
         int percDone = fe.getPercentDone();
         if (status == null)
             status = (percDone == 100) ? "Done" :  //Else
                                          ""; //Description was not available.
         statusBar.setText(status);  //A JLabel object part of the UI.
         progressBar.setValue(percDone); //A JProgressBar object part of the UI.
     }
     public void onEnd() { //Called right before any of the handleXXX methods.
         progressBar.setVisible(false);  //Because the computation has finished.
     }
     Public void handleResult(Object result) {  //Computation returned a result.
         //We have a non-null return value.  Cast it to what
         //loadSomeDataInTheBg() declared to return.
         SomeData data = (SomeData) result;

         //Update model, UI, etc.
     }
     public void handleCancellation() {  //Computation was cancelled.
         UserNotifier un = registry.getUserNotifier();
         un.notifyInfo("Data Loading", "SomeData task cancelled.");
     }
     Public void handleException(Throwable exc) {  //An error occurred.
         UserNotifier UN = registry.getUserNotifier();
         un.notifyError("Data Loading Failure",
                        "Couldn't retrieve SomeData.", exc);
```

```
    }
};
```

Note that the `observer`'s code in the example above works just like any other *Swing* listener. In fact, all events are delivered sequentially and within the *Swing* event dispatching thread. This means the `observer` can run synchronously with respect to the UI and will not need to worry about concurrency issues – as long as it runs within *Swing*. Finally, also note that subsequent feedback events imply computation progress and the `DSCallOutcomeEvent` is always the last event to be delivered in order of time.

The xxxLoader classes in agents.treeviewer are a good place to look at and see how to use data services view.

Execution

The next diagram analyzes a concrete call to a view to exemplify the pattern followed by all asynchronous calls in the various views. The call is mapped onto a command, the command is transferred to a processor for asynchronous execution, a handle to the call is returned to to invoker.

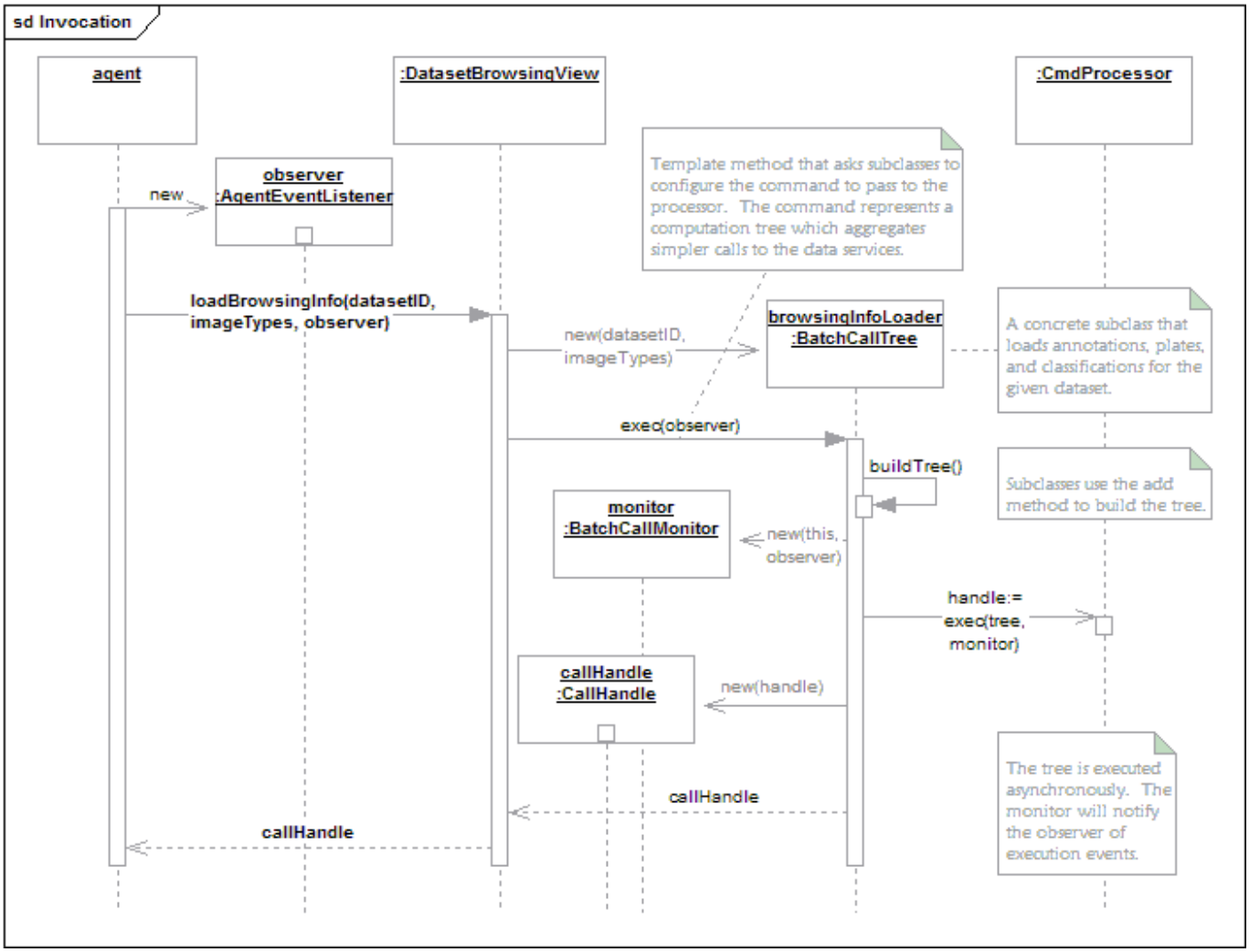


Figure 16.10: Retrieving data invocation

Initialization

The `DataViewsFactory` (in `env.data.views`) needs to be initialized before any concrete `BatchCallTree` (in `env.data.views`) is created. The reason for this is that `BatchCallTree`'s constructor needs to cache a reference to the registry so that concrete subclasses can access it later. The `DataViewsFactory` takes care of this initialization task during

the container's start-up procedure by calling `DataViewsFactory.initialize(Container)`. Any data service view should be created in `env.data.views` and declared in `DataViewsFactory.makeNew(Class)`. The method returns an implementation of the corresponding view.

See also:

Directory contents

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.10 Organization

The source code is organized as follows. All classes share a common base namespace:

```
org.openmicroscopy.shoola
```

Two main packages sit under the shoola directory:

- `agents`: All the classes related to concrete agents.
- `env`: All the classes that make up the runtime environment, that is the container.

The `agents` package is further broken down into:

- `events`
- `dataBrowser`
- `editor`
- `imviewer`
- `measurement`
- `metadata`
- `treeviewer`
- `util`

These packages contain the code for the Data Browser, Data Manager, Editor, Viewer, and Measurement agents. The events package contains the events that are used by all these agents.

The `env` package is also broken down into further sub-packages:

- `config`: Registry-related classes.
- `data`: Defines the client's side interface and implementation of the Remote Facade that we use to access the OMERO server.
- `event`: The event bus classes.
- `init`: Classes to perform initialization tasks at start-up.
- `log`: Adapter classes to wrap log4j[15].
- `cache`: Adapter classes to wrap ehcache[16].
- `rnd`: The image data provider.
- `ui`: The top frame window and the user notification widgets.

Note: Two extra packages are part of the project for convenience reason only:

- `svc`: Provides general services e.g. transport service using *HTTP*.
- `util`: Collection of classes that be used outside the OMERO structure

[15] http://logging.apache.org/log4j/
[16] http://ehcache.org/

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

16.11 Taskbar

The container provides a top-level window, the taskbar, to let users control some of the container's tasks like the connection to the remote service or quitting the application. The taskbar contains a menu bar and several toolbars. Agents can add their own entries to the menu bar and to the toolbars in order to trigger agent-specific actions, or can ignore the taskbar altogether. For example, agents that are UIs typically add an entry to the Window menu and to the Quick-launch toolbar (this should be done during the agent's linking phase) for top-level windows that the user can bring up. Some utility classes provide agents with functionality to link their top-level windows to the taskbar and to manage the display of those windows on screen.

16.11.1 Structure and dynamics

The following diagram shows the classes that provide the taskbar service and their relationships:

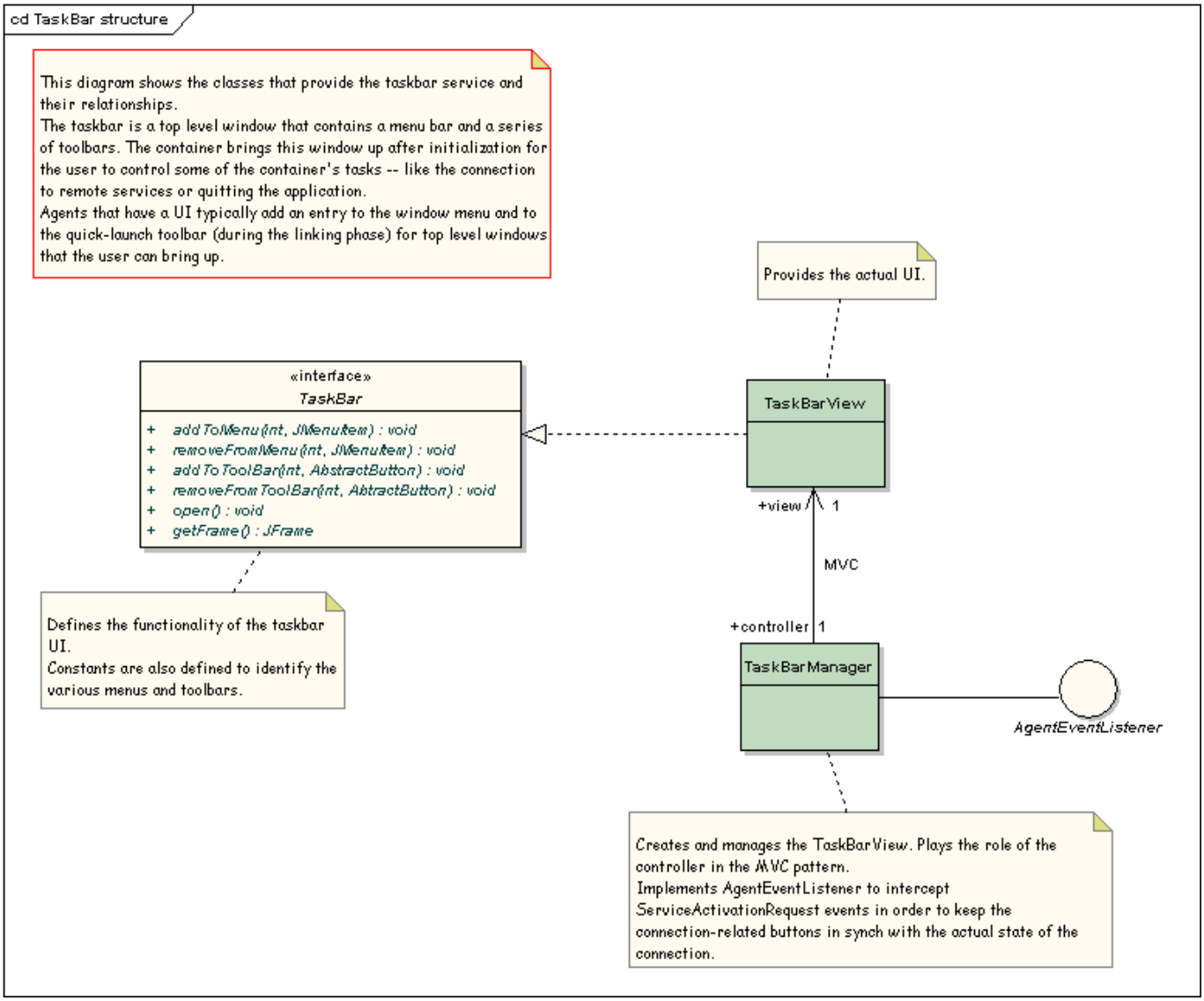


Figure 16.11: Taskbar structure

The following diagram shows how some utility classes that can be used to link windows to the `TaskBar` and to manage their display on screen:

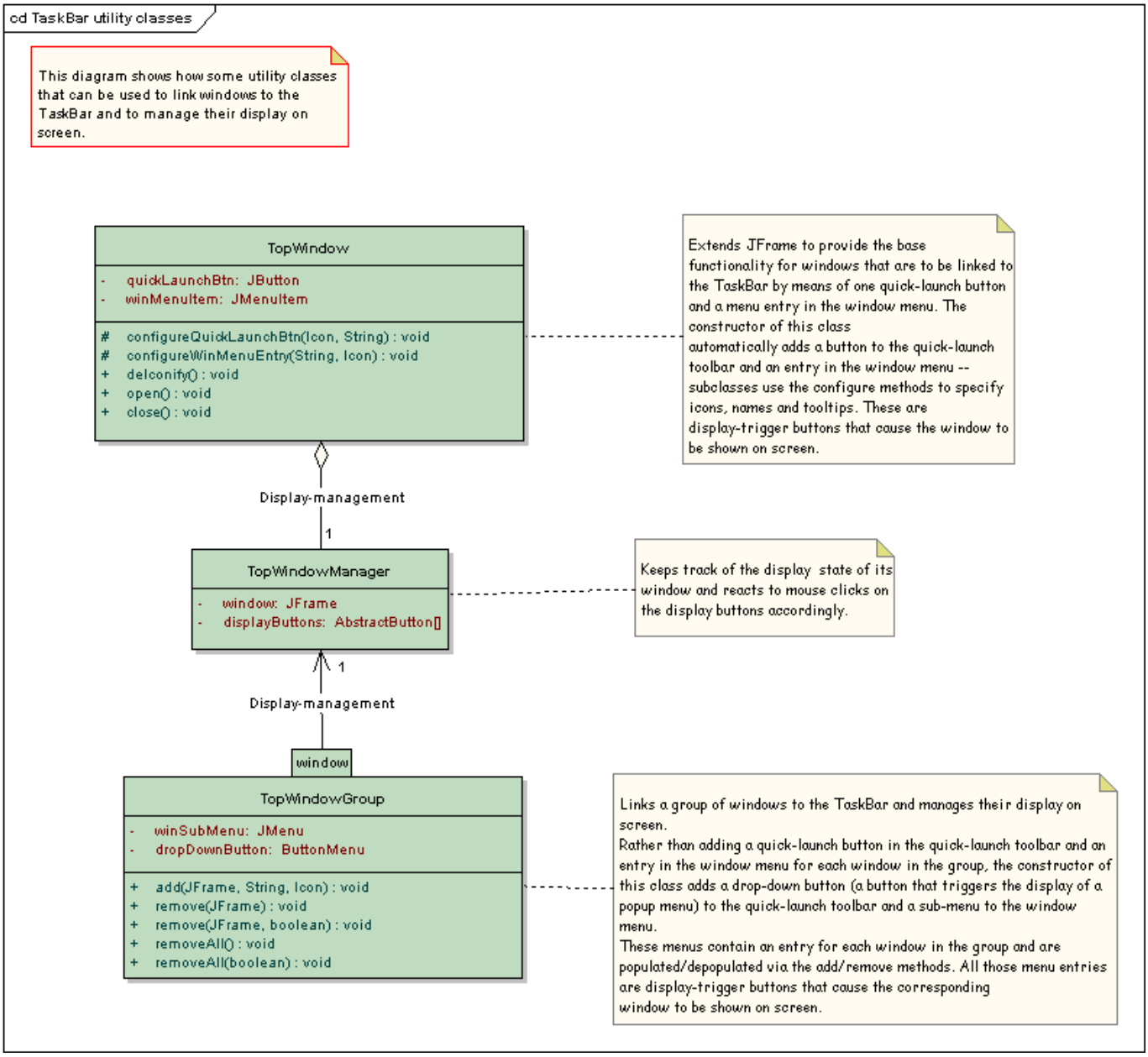


Figure 16.12: Taskbar utility classes

The following diagram shows how the display state of a top window is managed by the `TopWindowManager`:

16.11.2 How to

Agents can use the taskbar directly to add entries to the various menus and toolbars. After retrieving the `TaskBar` from the `Registry`, the `addXXX` and `removeXXX` are available to do the job.

Agents can add entries to any of the menus within the menu bar – `File`, `Connect`, `Tasks`, `Window`, `Help`. Two toolbars, `Tasks` and `Quick-Launch`, are also provided for agents to plug in their buttons. Buttons in the Tasks toolbar are usually shortcuts to entries in the Tasks menu. Similarly, buttons in the Quick-Launch toolbar are usually shortcuts to entries in the Window menu.

However, some utility classes provide agents with built-in functionality to link their top-level windows to the taskbar and to manage the display of those windows on screen.

The `TopWindow` class extends `JFrame` to provide the base functionality for windows that are to be linked to the `TaskBar` by means of one quick-launch button and a menu entry in the `Window` menu. The constructor of this class automatically adds a button to the `Quick-Launch` toolbar and an entry in the `Window` menu – subclasses use the `configureXXX` methods to

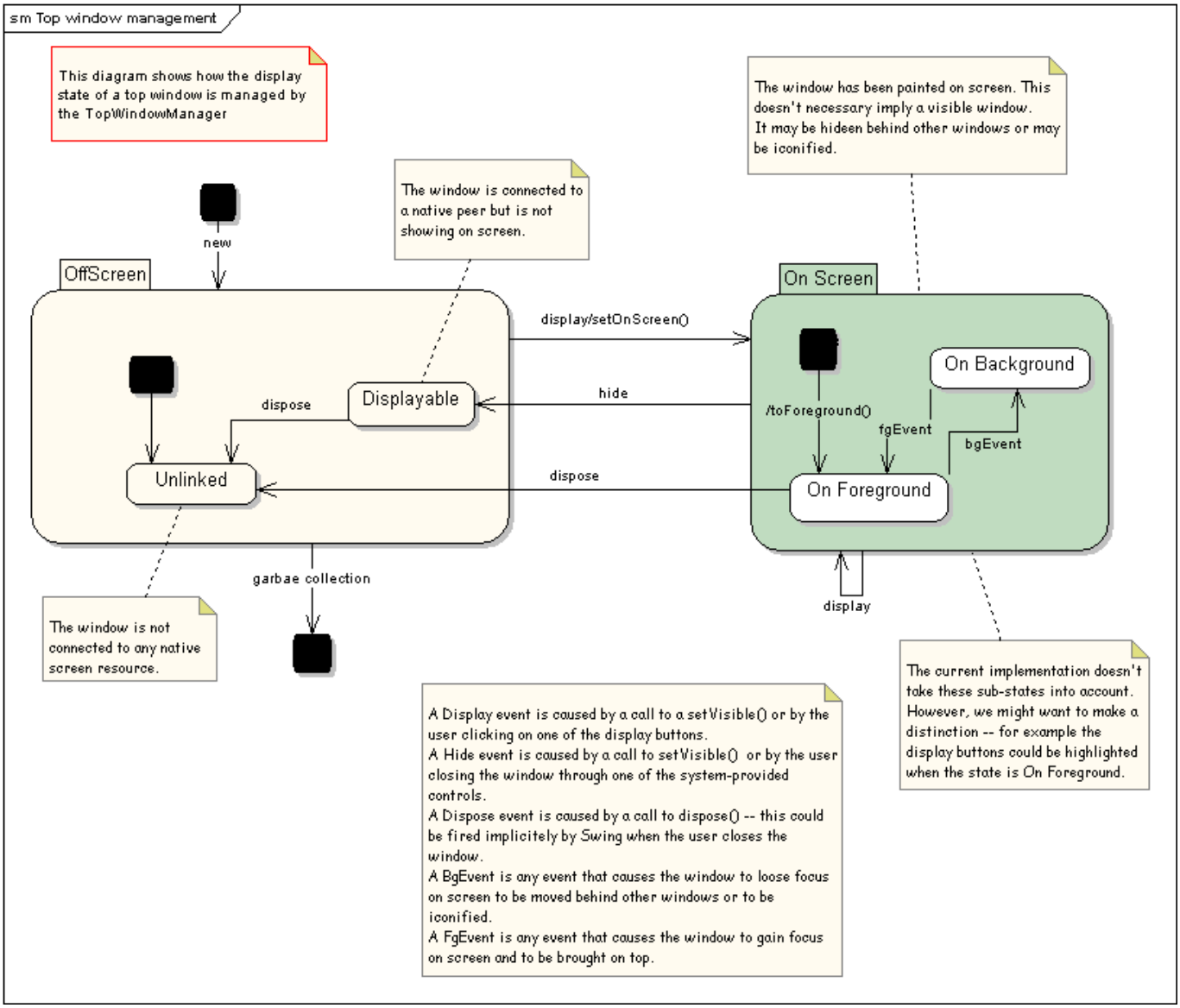


Figure 16.13: Taskbar window management

specify icons, names, and tool tips. These are display-trigger buttons that cause the window to be shown on screen. This class uses the `TopWindowManager` to control mouse clicks on these buttons as well as to manage the display state of the window – how this display state is managed is specified by the `TopWindowManager` state machine, which is represented in one of the previous diagrams.

Here is an example of a window that inherits from `TopWindow`:

```
class MainWindow
    extends TopWindow
{
    //Member fields omitted.

    //Specifies names, icons, and tool tips for the quick-launch button and the
    //window menu entry in the taskbar.
    private void configureDisplayButtons()
    {
        configureQuickLaunchBtn(icons.getIcon("agent.png"),
                                "Display the main window.");
        configureWinMenuEntry("Example Agent", icons.getIcon("agent.png"));
    }
```

```
    //Builds and lays out this window.
    private void buildGUI() { /* Omitted. */ }




    //Creates a new instance.
    MainWindow(Registry config)
    {
        //We have to specify the title of the window to the superclass
        //constructor and pass a reference to the TaskBar, which we get
        //from the Registry.
        super("Example Agent", config.getTaskBar());

            configureDisplayButtons();
            buildGUI();
    }

}
```

The `TopWindowGroup` class links a group of windows to the `TaskBar` and manages their display on screen. Rather than adding a quick-launch button in the `Quick-Launch` toolbar and an entry in the window menu for each window in the group, the constructor of this class adds a drop-down button (a button that triggers the display of a popup menu) to the `Quick-Launch` toolbar and a sub-menu to the `Window` menu. These menus contain an entry for each window in the group and are populated/depopulated via the `add/remove` methods. All those menu entries are display-trigger buttons that cause the corresponding window to be shown on screen. This class uses the `TopWindowManager` to control mouse clicks on these buttons as well as to manage the display state of each window in the group.

The following `UIManager` class provides an example of how to use the `TopWindowGroup` class. This example class creates and controls an instance of `MainWindow` (which we have already seen in the previous example) as well as `AuxiliaryWindow` instances. This latter class is just a window which contains two buttons and its code is omitted. `UIManager` delegates to the `TopWindowGroup` class the linkage of `AuxiliaryWindow`'s to the `TaskBar` as well as the management of their display state.

Follows the code:

```
class UIManager
{

    //Inherits from TopWindow, so it is automatically linked to the TaskBar.
    //Contains a button that we listen to.  When a mouse click occurs we call
    //createAuxWin().
    private MainWindow      mainWindow;

    //Manages all the AuxiliaryWindow's that we have created and not destroyed yet.
    private TopWindowGroup      auxWinGroup;

    //Counts how many {@link AuxiliaryWindow}'s that we have created so far.
    private int         auxWinCount;

    //Cached reference to access the icons.
    private IconFactory     icons;


    //Creates a new instance.
    UIManager(Registry config)
    {

        auxWinCount = 0;
        icons = (IconFactory) config.lookup("/resources/icons/MyFactory");
        mainWindow = new MainWindow(config);
```

```
        //The MainWindow contains a button (not shown in the previous example)
        //which we listen to in order to trigger the creation of new
        //AuxiliaryWindow's.
        mainWindow.openAuxiliary.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) { createAuxWin(); }
        });




        //We now create the window group.  The text we pass will be displayed by
        //the sub-menu within the Window menu along with the icon, which will also
        //be the icon displayed by the drop-down button in the Quick-Launch
        //toolbar.
        auxWinGroup = new TopWindowGroup("Aux Win",
                    icons.getIcon("edu_languages.png"),
                    config.getTaskBar());

    }

    //Creates an AuxiliaryWindow and adds it to the auxWinGroup.
    //Every AuxiliaryWindow contains two buttons, one labeled "Close" and the other
    //"Dispose".  We listen to mouse clicks on these buttons in order to hide the
    //window when the "Close" button is clicked and to remove the window (and dispose
    //of it) from the auxWinGroup when the "Dispose" button is clicked.
    private void createAuxWin()
    {
        String title = "Aux Window "+(++auxWinCount);
        final AuxiliaryWindow aw = new AuxiliaryWindow(title);

        //Attach listeners and specify actions.
        aw.close.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {aw.setVisible(false);}
        });
        aw.dispose.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                //Remove from group and dispose.
                auxWinGroup.remove(aw, true);
            }
        });

        //Add to the group.  An entry will be added both to the Window sub-menu
        //and to the popup menu triggered by the drop-down button in the
        //Quick-Launch toolbar.  We set the display text of those entries to be
        //the same as the window's title, but we don't specify any icon.
        auxWinGroup.add(aw, title, null);

        //Bring the window up.
        aw.open();
    }

    //Releases all UI resources currently in use and returns them to the OS.
    void disposeUI()
    {
        mainWindow.dispose();
        auxWinGroup.removeAll(true);  //Empty group and dispose of all windows.
    }

}
```

CHAPTER

SEVENTEEN

MORE ON API USAGE

OMERO can be extended by modifying these clients or by writing your own in any of the supported languages.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

17.1 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[1] **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: examples[2] and buildable/runnable via scons[3]:

```
cd omero-src
./build.py build-all
cd omero-src/examples
python ../target/scons/scons.py
```

Other examples (in Python) can be found *here*.

17.1.1 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.

17.1.2 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[4] from ZeroC. Ice is comparable to CORBA in many ways, but is typically easier to use. For ZeroC's comparison of Ice to CORBA, see iceVsCorba.html[5].

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

[1] http://zeroc.com/doc/Ice-3.3.0/manual/Hello.html#22064
[2] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/examples
[3] http://www.scons.org
[4] http://www.zeroc.com
[5] http://zeroc.com/iceVsCorba.html

fallacies of distributed computing[6] 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.

17.1.3 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)[7] 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 components/blitz/resources/omero/System.ice[8]:

```
// 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 components/blitz/resources/omero/Collections.ice[9]. 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 components/blitz/resources/omero/Constants.ice[10]. 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".

[6] http://en.wikipedia.org/wiki/Fallacies_of_Distributed_Computing
[7] http://en.wikipedia.org/wiki/Interface_description_language
[8] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/resources/omero/System.ice
[9] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/resources/omero/Collections.ice
[10] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/resources/omero/Constants.ice

```
#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 << MESSAGESIZEMAX << "kb" << std::endl;
    std::cout << "By default, client.createSession() will wait ";
    std::cout << (CONNECTTIMEOUT / 1000) << " seconds for a connection" << std::endl;
}
```

Example: examples/OmeroClients/constants.cpp[11]

```
sz=omero.constants.MESSAGESIZEMAX.value;
to=omero.constants.CONNECTTIMEOUT.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: examples/OmeroClients/constants.m[12]

```
from omero.constants import *
print "By default, no method call can pass more than %s kb" % MESSAGESIZEMAX
print "By default, client.createSession() will wait %s seconds for a connection" % (CONNECTTIMEOUT/1000
```

Example: examples/OmeroClients/constants.py[13]

```
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.MESSAGESIZEMAX.value));
        System.out.println(String.format(
            "By default, client.createSession() will wait %s seconds for a connection",
            omero.constants.CONNECTTIMEOUT.value/1000));
    }
}
```

Example: examples/OmeroClients/constants.java[14]

- **Enums.** Finally, enumerations which are less used through *OMERO Application Programming Interface*, but which can be useful for simplyifying 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 << MAXIMUMINTENSITY;
    std::cout << std::endl;
}
```

Example: examples/OmeroClients/enumerations.cpp[15]

[11] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/constants.cpp
[12] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/constants.m
[13] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/constants.py
[14] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/constants.java
[15] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/enumerations.cpp

```
v=omero.constants.projection.ProjectionType.MAXIMUMINTENSITY.value();
disp(sprintf('IProjection takes arguments of the form: %s', v));
```

Example: examples/OmeroClients/enumerations.m[16]

```
import omero
import omero_Constants_ice
print "IProjection takes arguments of the form: %s" % omero.constants.projection.ProjectionType.MAXIMUM
```

Example: examples/OmeroClients/enumerations.py[17]

```
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.MAXIMUMINTENSITY));
    }
}
```

Example: examples/OmeroClients/enumerations.java[18]

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[19] 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: examples/OmeroClients/primitives.cpp[20]

```
import omero.rtypes;
a = rtypes.rstring('value');
b = rtypes.rbool(true);
l = rtypes.rlong(1);
i = rtypes.rint(1);
```

Example: examples/OmeroClients/primitives.m[21]

[16] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/enumerations.m
[17] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/enumerations.py
[18] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/enumerations.java
[19] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/resources/omero/RTypes.ice
[20] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/primitives.cpp
[21] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/primitives.m

```
from omero.rtypes import *
s = rstring("value")
b = rbool(True)
l = rlong(1)
i = rint(1)
```

Example: examples/OmeroClients/primitives.py[22]

```
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(1l);
        omero.RInt i = rint(1);
    }
}
```

Example: examples/OmeroClients/primitives.java[23]

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: examples/OmeroClients/rcollection.cpp[24]

```
% 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: examples/OmeroClients/rcollection.m[25]

```
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: examples/OmeroClients/rcollection.py[26]

[22] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/primitives.py
[23] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/primitives.java
[24] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/rcollection.cpp
[25] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/rcollection.m
[26] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/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: examples/OmeroClients/rcollection.java[27]

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 "**r**emote **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[28] 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: examples/OmeroClients/details.cpp[29]

[27] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/rcollection.java
[28] http://www.openmicroscopy.org/site/support/ome-model/ome-xml/
[29] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/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: examples/OmeroClients/details.m[30]

```
import omero
import omero.clients
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: examples/OmeroClients/details.py[31]

```
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();
```

[30] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/details.m
[31] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/details.py

```
    }
}
```

Example: examples/OmeroClients/details.java[32]

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: examples/OmeroClients/constructors.cpp[33]

```
import omero.model.*;
image = ImageI();
dataset = DatasetI(1, false);
image.linkDataset(dataset)
```

Example: examples/OmeroClients/constructors.m[34]

```
import omero
import omero.clients
image = omero.model.ImageI()
dataset = omero.model.DatasetI(long(1), False)
image.linkDataset(dataset)
```

Example: examples/OmeroClients/constructors.py[35]

```
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);
```

[32] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/details.java
[33] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/constructors.cpp
[34] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/constructors.m
[35] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/constructors.py

```
    }
}
```

Example: examples/OmeroClients/constructors.java[36]

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: examples/OmeroClients/iterators.cpp[37]

```
import omero.model.*;
image = ImageI();
dataset = DatasetI();
link = dataset.linkImage(image);
it = image.iterateDatasetLinks();
while it.hasNext()
   it.next().getChild().getName()
end
```

Example: examples/OmeroClients/iterators.m[38]

```
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: examples/OmeroClients/iterators.py[39]

[36] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/constructors.java
[37] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/iterators.cpp
[38] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/iterators.m
[39] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/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 {
    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: examples/OmeroClients/iterators.java[40]

]

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: examples/OmeroClients/staticfields.cpp[41]

```
disp(omero.model.ImageI.NAME);
disp(omero.model.ImageI.DATASETLINKS);
```

Example: examples/OmeroClients/staticfields.m[42]

```
import omero
from omero_model_ImageI import ImageI as ImageI
print ImageI.NAME
print ImageI.DATASETLINKS
```

Example: examples/OmeroClients/staticfields.py[43]

```
import omero.model.ImageI;
public class staticfields {
    public static void main(String[] args) {
        System.out.println(ImageI.NAME);
        System.out.println(ImageI.DATASETLINKS);
    }
}
```

Example: examples/OmeroClients/staticfields.java[44]

[40] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/iterators.java
[41] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/staticfields.cpp
[42] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/staticfields.m
[43] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/staticfields.py
[44] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/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: examples/OmeroClients/unloaded.cpp[45]

```
image = omero.model.ImageI();                % A loaded object by default
assert(image.isLoaded());
image.unload();
assert( ~ image.isLoaded() );                % can then be unloaded
image = omero.model.ImageI( 1, false );
assert( ~ image.isLoaded() );                % Creates an unloaded "proxy"
image.getId();                               % Ok.
try
    image.getName();                         % No data access is allowed other than id
catch ME
    % OK
end
```

Example: examples/OmeroClients/unloaded.m[46]

```
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())
image.getId()                               # Ok
try:
    image.getName()                         # No data access is allowed other than id.
except:
    pass
```

[45] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/unloaded.cpp
[46] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/unloaded.m

Example: examples/OmeroClients/unloaded.py[47]

```
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: examples/OmeroClients/unloaded.java[48]

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[]) {
    ImagePtr 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);
    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
```

[47] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/unloaded.py
[48] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/unloaded.java

```
    // 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: examples/OmeroClients/collectionmethods.cpp[49]

```
import omero.model.*;
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
```

[49] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/collectionmethods.cpp

```
% 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: examples/OmeroClients/collectionmethods.m[50]

```
import omero
import omero.clients
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) )
```

[50] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/collectionmethods.m

```
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: examples/OmeroClients/collectionmethods.py[51]

```
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);
```

[51] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/collectionmethods.py

```
        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: examples/OmeroClients/collectionmethods.java[52]

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.

- IObject[53] is the root interface for all object types. **Methods**: `getId()`, `getDetails()`, …
- IEnum[54] is an enumeration value. **Methods**: `getValue()`
- ILink[55] is a link between two other types. **Methods**: `getParent()`, `getChild()`
- IMutable[56] 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() );
```

[52] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/collectionmethods.java
[53] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/src/ome/model/IObject.java
[54] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/src/ome/model/IEnum.java
[55] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/src/ome/model/ILink.java
[56] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/src/ome/model/IMutable.java

```
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())
    % deleteAnnotations(someObject);
end
```

Example: examples/OmeroClients/interfaces.m[57]

```
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: examples/OmeroClients/interfaces.py[58]

```
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: examples/OmeroClients/interfaces.java[59]

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

[57] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/interfaces.m
[58] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/interfaces.py
[59] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/interfaces.java

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: examples/OmeroClients/smartpointers.cpp[60]

```
# 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[61] 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);
```

[60] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/smartpointers.cpp
[61] http://zeroc.com/doc/Ice-3.3.0/manual/Cpp.7.14.html

```
    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: examples/OmeroClients/clientpointer.cpp[62]

```
# 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"
#
# With __getattr__ and __setattr__
```

[62] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/clientpointer.cpp

```
#
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: examples/OmeroClients/getsetattr.py[63]

```
# 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
>>>
```

17.1.4 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,

[63] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/getsetattr.py

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?
    }
    // 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: examples/OmeroClients/configuration.cpp[64]

```
% 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
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: examples/OmeroClients/configuration.m[65]

```
import omero
import Ice
# All configuration in file pointed to by
# --Ice.Config=file.config or ICE_CONFIG
# environment variable;
```

[64] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/configuration.cpp
[65] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/configuration.m

```
# 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:
    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: examples/OmeroClients/configuration.py[66]

```
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 {
```

[66] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/configuration.py

```
        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();
    }
    // 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: examples/OmeroClients/configuration.java[67]

To find out more about using the `Ice.Config` file for configuration, see etc/ice.config[68].

[67] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/configuration.java
[68] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/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 components/blitz/resources/omero[69]. The central definitions are in components/blitz/resources/omero/API.ice[70], 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

[69] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/blitz/resources/omero
[70] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/resources/omero/API.ice

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: examples/OmeroClients/timeout.py[71]

```
import omero.*;
import omero.api.*;
import omero.model.*;
import omero.sys.*;
public class timeout {
    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
```

[71] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/timeout.py

```
                    }
                    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: examples/OmeroClients/timeout.java[72]

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.

17.1.5 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;
```

[72] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/timeout.java

```
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: examples/OmeroClients/queries.cpp[73]

```
[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%'));
    results = q.findAllByQuery(query_string, p) % java.util.List
catch ME
    client.closeSession();
end
```

Example: examples/OmeroClients/queries.m[74]

```
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: examples/OmeroClients/queries.py[75]

```
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 {
```

[73] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/queries.cpp
[74] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/queries.m
[75] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/queries.py

```
            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: examples/OmeroClients/queries.java[76]

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.

17.1.6 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: examples/OmeroClients/updates.cpp[77]

```
[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
```

[76] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/queries.java
[77] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/updates.cpp

Example: examples/OmeroClients/updates.m[78]

```
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")
    i.acquisitionDate = rtime(time.time() * 1000)
    sf = client.createSession()
    u = sf.getUpdateService()
    i = u.saveAndReturnObject( i )
finally:
    client.closeSession()
```

Example: examples/OmeroClients/updates.py[79]

```
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: examples/OmeroClients/updates.java[80]

17.1.7 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
```

[78] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/updates.m
[79] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/updates.py
[80] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/updates.java

```
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: examples/TreeList/AllProjects.m[81]

```
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
            ParametersI().add("name", rstring(username)))
```

Example: examples/TreeList/AllProjects.py[82]

```
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
            new ParametersI().add("name", rstring(username)));
        return (List<Project>) rv;
    }
}
```

Example: examples/TreeList/AllProjects.java[83]

```
# 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
```

[81] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/TreeList/AllProjects.m
[82] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/TreeList/AllProjects.py
[83] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/TreeList/AllProjects.java

Example: examples/TreeList/PrintProjects.m[84]

```
def print_(projects):
    for project in projects:
        print project.getName().val
        for pdl in project.copyDatasetLinks():
            dataset = pdl.getChild()
            print "  " + dataset.getName().val
```

Example: examples/TreeList/PrintProjects.py[85]

```
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: examples/TreeList/PrintProjects.java[86]

```
#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->getQueryServ
        PrintProjects::print(projects);
    } catch (...) {
        client.closeSession();
    }
    return rc;
}
```

Example: examples/TreeList/Main.cpp[87]

[84] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/TreeList/PrintProjects.m
[85] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/TreeList/PrintProjects.py
[86] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/TreeList/PrintProjects.java
[87] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/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: examples/TreeList/Main.m[88]

```
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: examples/TreeList/Main.py[89]

```
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);
        try {
            ServiceFactoryPrx factory = client.createSession(user, pass);
            List<Project> projects = AllProjects.getProjects(factory.getQueryService(), user);
            PrintProjects.print(projects);
        } finally {
            client.closeSession();
```

[88] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/TreeList/Main.m
[89] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/TreeList/Main.py

```
        }
    }
}
```

Example: examples/TreeList/Main.java[90]

17.1.8 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
    omero::api::ServiceFactoryPrx sudoSf = sudoClient->joinSession( sudoSession->getUuid()->getValue()
    omero::api::IAdminPrx sudoAdminSvc = sudoSf->getAdminService();
    std::cout << sudoAdminSvc->getEventContext()->userName;
}
```

Example: examples/OmeroClients/sudo.cpp[91]

```
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(sudoAdmin.Svc.getEventContext().userName);
catch ME
```

[90] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/TreeList/Main.java
[91] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/sudo.cpp

```
    sudoClient.closeSession();
    client.closeSession();
end
```

Example: examples/OmeroClients/sudo.m[92]

```
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: examples/OmeroClients/sudo.py[93]

```
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);
        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
            ServiceFactoryPrx sudoSf = sudoClient.joinSession( sudoSession.getUuid().getValue() );
            IAdminPrx sudoAdminSvc = sudoSf.getAdminService();
            System.out.println( sudoAdminSvc.getEventContext().userName );
        } finally {
            sudoClient.closeSession();
            client.closeSession();
        }
    }
}
```

Example: examples/OmeroClients/sudo.java[94]

[92] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/sudo.m
[93] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/examples/OmeroClients/sudo.py
[94] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/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`

17.1.9 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[95].

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[96] 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.
- Python applications can segfault under certain orderings of imports: See Bus Error under Mac OX 10.4 and IcePy 3.3.0[97]

[95] https://github.com/openmicroscopy/openmicroscopy
[96] http://ci.openmicroscopy.org/
[97] http://zeroc.com/forums/bug-reports/3883-bus-error-under-mac-ox-10-4-icepy-3-3-0-a.html

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

17.2 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:

```
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[98] has methods for creating Stateless and Stateful services (see below).
  - Stateless services are obtained using "get…" methods e.g. `getQueryService()`
  - Stateful services are obtained using "create…" methods e.g. `createRenderingEngine()`
- Services will provide access to omero.model.objects. You will then need the API for these objects[99], e.g. Dataset, Image, Pixels etc.

17.2.1 Services list

The ome.api[100] 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[101], API[102] for working with Experimenters, Groups and the current Context (switching groups etc).
- ConfigService: src[103], API[104] for getting and setting config parameters.
- ContainerService: API[105] for loading Project, Dataset and Image hierarchies.
- DeleteService: API[106] for deleting objects asynchronously (delete queue).
- LdapService: src[107], API[108] for communicating with LDAP servers.
- MetadataService: API[109] for working with Annotations.
- PixelsService: API[110] for pixels stats and creating Images with existing or new Pixels.
- ProjectionService API[111]
- QueryService: src[112], API[113] for custom SQL-like queries.
- RenderingSettingsService API[114] for copying, pasting & resetting rendering settings.

[98] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/ServiceFactory.html
[99] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/model.html
[100] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/common/src/ome/api
[101] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IAdmin.java
[102] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IAdmin.html
[103] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IConfig.java
[104] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IConfig.html
[105] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IContainer.html
[106] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IDelete.html
[107] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/ILdap.java
[108] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/ILdap.html
[109] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IMetadata.html
[110] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IPixels.html
[111] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IProjection.html
[112] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IQuery.java
[113] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/ome/api/IQuery.html
[114] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IRenderingSettings.html

- RepositoryInfo API[115] disk space stats.
- RoiService API[116] working with ROIs.
- ScriptService API[117] for uploading and launching Python scripts.
- SessionService API[118] for creating and working with OMERO sessions.
- ShareService API[119]
- TimelineService API[120] for queries based on time.
- TypesService API[121] for Enumerations.
- UpdateService: src[122], API[123] for saving and deleting omero.model objects.

Service Level 2

- IContainer[124]
- ITypes[125]

Stateful/Binary Services

- RawFileStore: src[126], API[127]
- RawPixelsStore: src[128], API[129]
- RenderingEngine: src[130], API[131] (see *OMERO rendering engine* for more)
- ThumbnailStore: src[132], API[133]
- IScale[134]

A complete list of service APIs can be found here[135] and some examples of API use in Python are provided. Java or C++ code can use the same API in a very similar manner.

17.2.2 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 QueryParemeters which are explained under *Queries*. 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[136] 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.

[115] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IRepositoryInfo.html
[116] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IRoi.html
[117] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IScript.html
[118] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/ISession.html
[119] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/IShare.html
[120] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/ITimeline.html
[121] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/ITypes.html
[122] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IUpdate.java
[123] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/ome/api/IUpdate.html
[124] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IContainer.java
[125] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/ITypes.java
[126] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/RawFileStore.java
[127] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/ome/api/RawFileStore.html
[128] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/RawPixelsStore.java
[129] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/ome/api/RawPixelsStore.html
[130] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/omeis/providers/re/RenderingEngine.java
[131] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api/RenderingEngine.html
[132] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/ThumbnailStore.java
[133] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/ome/api/ThumbnailStore.html
[134] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IScale.java
[135] http://ci.openmicroscopy.org/job/OMERO-trunk/javadoc/slice2html/omero/api.html
[136] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/src/ome/model/IObject.java

Dirty checks follow the Three Commandments:

1. Any IObject-valued field with unloaded set to true is treated as a place holder (proxy) and is re-loaded 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[137] interface defines all the actions necessary to administer the *Server security and firewalls* . It is explained further on the *OMERO admin interface* page.

Pojos

Certain operations, like those dealing with data management and viewing, happen more frequently than others (like defining microscopes). Those have been collected in the IContainer[138] interface. IContainer simplifies a few very common queries, and there is a related package ("pojos.*") for working with the returned graphs. OMERO.insight works almost exclusively with the IContainer interface for its non-binary needs.

17.2.3 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'"
```

17.2.4 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.create\*` methods). Once obtained, a stateful service proxy can only be used by a single user. After task completion, the service should be closed (`proxy.close()`) to free up server resources.

17.2.5 How to write a service

A tutorial is available at *How To create a service*. In general, if a properly annotated service is placed in any JAR of the OMERO EAR file (see *Build System* for more) then the service will be deployed to the server. In the case of *OMERO.blitz*, the service must be properly defined under components/blitz/resources[139].

[137] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IAdmin.java
[138] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IContainer.java
[139] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/blitz/resources

17.2.6 OMERO annotations for validation

The server-side implementation of these interfaces makes use of ((JDK5)) *Structured annotations* and an *AOP* interceptor to validate all method parameters. Calls to `pojos.findContainerHierarches` 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.findContainerHierarches(Class,Set,Map)
```

is implemented as

```
pojos.findContainerHierarchies(@NotNull Class, @NotNull @Validate(Integer.class) Set, Map)
```

See also:

Queries, *OMERO rendering engine*, *Exception handling*

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

17.3 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 `@javax.ejb.security.RolesAllowed` annotations on the LocalAdmin[140] class define who can use which methods.

17.3.1 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 );
```

[140] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/api/local/LocalAdmin.java

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`.

17.3.2 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.

17.3.3 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

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

17.4 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[141] 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[142]).

17.5 Delete behavior (technical)

Configuring what gets deleted is done using an XML file. The technical specification of delete behavior can be found components/server/resources/ome/services/spec.xml[143]

[141] http://trac.openmicroscopy.org.uk/ome/ticket/2615
[142] http://trac.openmicroscopy.org.uk/ome/ticket/2911
[143] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/spec.xml

17.5.1 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.

17.5.2 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.

17.5.3 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.

17.5.4 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.

17.5.5 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 *Permissions overview* 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.

17.5.6 Group owner rights

An owner of the group, usually a PI, can delete anything that belongs to other members of the group.

17.5.7 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.

- Annotations of annotations are not deleted, e.g. a Tag is not deleted if a Tag Set is deleted (only true if directly using the API).
- 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.

17.5.8 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 (see *Unix* and *Windows* versions). It is possible that some files may not be successfully deleted if they are locked for any reason. This is a known problem on Windows servers. In this case, the undeleted files can be removed manually via `bin/omero admin cleanse`

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

17.6 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*
- *The Command Line Import* tool

17.6.1 Components

The primary classes which make up the Import Library are:

- ImportLibrary.java[144] itself, which is the main driver
- ImportCandidates.java[145] which takes file paths and determines the proper files to import
- ImportConfig.java[146], an extensible mechanism for storing the properties used during import
- ImportEvent.java[147], the various events raised during import to `IObserver`and `IObservable` implementations
- OMEROMetadataStoreClient.java[148], the low-level connection to the server
- OMEROWrapper.java[149], the OMERO adapter for the Bio-Formats `ImageReaders` class
- In OMERO.insight, the main entry point is the importImage method of OMEROGateway.java[150]
- In the CLI, the main entry point is the CommandLineImporter[151] class

[144] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/src/ome/formats/importer/ImportLibrary.java
[145] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/src/ome/formats/importer/ImportCandidates.java
[146] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/src/ome/formats/importer/ImportConfig.java
[147] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/src/ome/formats/importer/ImportEvent.java
[148] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/src/ome/formats/OMEROMetadataStoreClient.java
[149] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/src/ome/formats/importer/OMEROWrapper.java
[150] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/insight/SRC/org/openmicroscopy/shoola/env/data/OMEROGateway.java
[151] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroImporter/src/ome/formats/importer/cli/CommandLineImporter.java

17.6.2 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

17.7 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[152]):

- The environment property setting `OMERO_TEMPDIR`
- 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()`

17.7.1 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.

17.7.2 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()`.

17.7.3 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

[152] http://trac.openmicroscopy.org.uk/ome/ticket/1653

```
TempFileManager.create_path("omero", ".tmp", true);
```

Note: All contents of the generated directory will be deleted.

See also:

#1534[153]

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

17.8 Exception handling

17.8.1 Client exceptions

The exceptions which can be received by a client due to a remote call on the OMERO server are all defined in components/blitz/resources/omero/ServerErrors.ice[154] (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="http://zeroc.com/doc/Ice-3.3.1/manual/Slice.5.10.html#50592"> Figure 4.4 </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:
 * ===============
 *  ServerError (root server exception)
 *    |
 *    |_ InternalException (server bug)
 *    |
 *    |_ ResourceError (non-recoverable)
```

[153] http://trac.openmicroscopy.org.uk/ome/ticket/1534
[154] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/resources/omero/ServerErrors.ice

```
*   |  \_ 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-extant session)
*      |_ SessionTimeoutException (session timed out; not yet removed)
*      \_ ShutdownInProgress      (session on this server will most likely be destroyed)
* </pre>
*
*
* <p>
* 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.
* </p>
*
*
* <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/resources/omero/Licens
*   |  |
*   |  \_ 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::UknownException (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
    {
    };
  /**
   * 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 expried. 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 ------------------------------
/**
 * 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 modifed. Clients may want to wait the given
 * backOff period, and retry.
 */
```

```
exception DatabaseBusyException extends ConcurrencyException
  {
  };
/**
 * Conflicting changes to the same piece of data.
 */
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
  /**
   * 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
```

17.8.2 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[155]). All exceptions thrown by code are caught in a `try {} catch (Throwable t) {}` block. Exceptions which do not subclass ome.conditions.RootException[156] are wrapped in an ome.conditions.InternalException[157].

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[158].

Hierarchy

The current exception hierarchy (package ome.conditions[159]) 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.

[155] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12//components/server/src/ome/services/util/ServiceHandler.java
[156] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/src/ome/conditions/RootException.java
[157] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/src/ome/conditions/InternalException.java
[158] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/services.xml
[159] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/model/src/ome/conditions

17.8.3 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?

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

17.9 Omero logging

All OMERO components written in Java use Log4J[160] (mostly via Commons-Logging[161]); 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.

17.9.1 Java clients

Java clients log to `$HOME/omero/log`. The number of files and their size are limited.

Logging is configured by log4j properties files contained within the jars themselves. For OmeroImport, the file is here: log4j.properties[162], which delegates to LogAppenderProxy[163] for much of the configuration.

Another file in that directory – log4j-cli.properties[164] 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.

17.9.2 Java servers

Java server components are configured by passing `-Dlog4j.configuration=etc/log4j.xml` to each Java process. Entry.java[165] guarantees that the *log4j.xml <etc/log4j.xml>* file is read periodically so that changes to your logging configuration do not require a restart.

By default, the output from log4j 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.

17.9.3 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 the templates.xml[166].

[160] http://logging.apache.org/
[161] http://commons.apache.org/logging/
[162] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroImporter/resources/log4j.properties
[163] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroImporter/src/ome/formats/importer/util/LogAppenderProxy.java
[164] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroImporter/resources/log4j-cli.properties
[165] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/src/ome/services/blitz/Entry.java
[166] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/templates.xml

All the "omero.logging.*" properties can be overwritten in your default.xml[167] file (or on Windows, etc/grid/windefault.xml[168]). See the "Profile" properties block for how to configure for your site.

Similar to log4j, logging is configured to be written to `var/log/<servername>.log` and to maintain 9 backups of at most 500MB.

17.9.4 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.

17.9.5 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[169] for more information).

17.9.6 Upcoming documentation on logging in Omero

- –debug / –report / –email / –upload
- tracing and warning settings
- zipping logs for feedback

[167] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/default.xml
[168] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/windefault.xml
[169] http://trac.openmicroscopy.org.uk/ome/ticket/1449

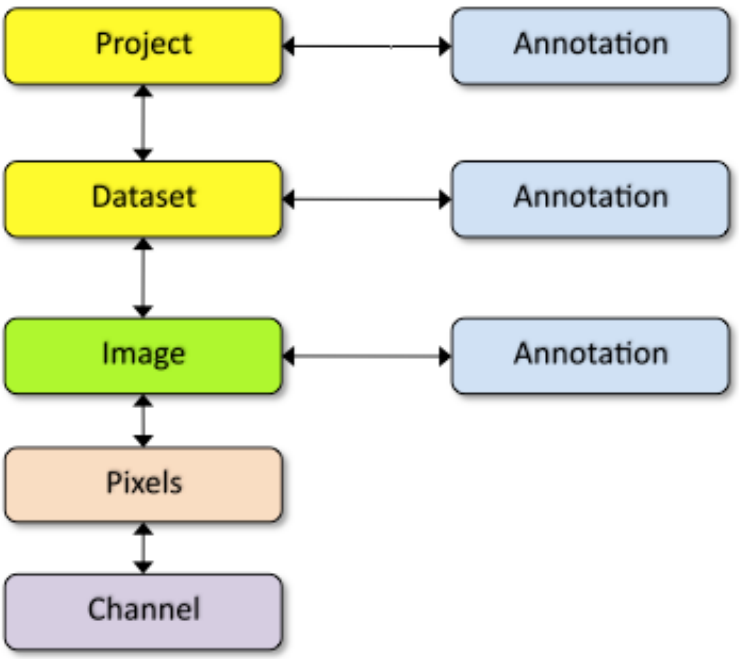
CHAPTER

EIGHTEEN

THE OME DATA MODEL

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

18.1 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[1] is the definition of a vocabulary for working with microscopic data. This vocabulary has a representation in the XML specification[2], in the database (the data model), and in code. This last representation is the object model with which we will concern ourselves here.

[1] http://www.openmicroscopy.org/site
[2] http://www.openmicroscopy.org/site/support/ome-model/ome-xml/

Because of its complexity, the object model is generated from a central definition[3] using our own code-generator[4]. It relies on no libraries and can be used in both the server and the RMI clients. *OMERO.blitz* uses a second mapping[5] to generate *OMERO Java language bindings*, *OMERO Python language bindings*, and *OMERO C++ language bindings* classes, which can be mapped[6] 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[7]. 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.**

18.1.1 OMERO type language

The *OME-Remote Objects* 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 at:

- components/model/resources/mappings/acquisition.ome.xml[8]

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.

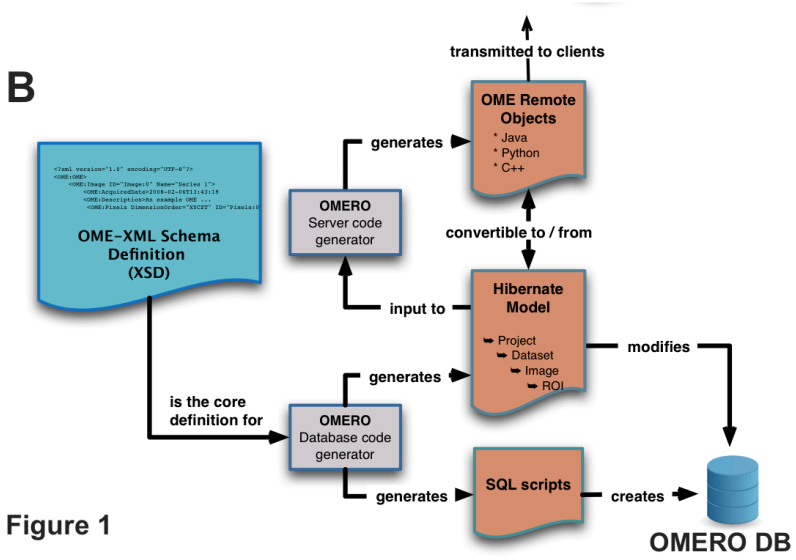


Figure 1

Conceptually, the XSD files under the components/specification[9] source directory are the starting point for all code generation. Currently however, the files under components/model/resources/mappings[10] are hand-written based on the XSD files.

The ant-task created from the components/dsl/src[11] Java files is then used to turn the mapping files into generated Java code under the file:*model/target/generated/src* directory. These classes are all within the ome.model package. A few hand-written Java classes can also be found in components/model/src/ome/model/internal[12].

[3] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/model
[4] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/dsl
[5] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/blitz/resources/templates
[6] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/src/omero/util/IceMapper.java
[7] http://www.hibernate.org
[8] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/resources/mappings/acquisition.ome.xml
[9] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/specification
[10] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/model/resources/mappings
[11] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/dsl/src
[12] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/model/src/ome/model/internal

The build-schema target takes the generate ome.model classes as input and generates the sql/psql[13] scripts which get used on `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 components/server[14] component.

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 "blitz"" layer. These classes are in the omero.model package. They are generated by another call to the DSL ant task in order to generate the Java, Python, C++, and Ice files under file:*components/blitz/generated*.

The generated Ice files along with the hand-written Ice files from components/blitz/resources/omero[15] 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

Properties

Mutable, annotated, global.

18.1.2 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.PixelsType,
       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"`.

[13] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/sql/psql
[14] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/server
[15] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/blitz/resources/omero

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
Device: 1602h/5634d     Inode: 376221      Links: 1
Access: (0600/-rw-------)  Uid: ( 1003/    josh)   Gid: ( 1001/ ome)
Access: 2006-01-25 20:40:30.000000000 +0100
Modify: 2006-01-25 20:40:30.000000000 +0100
Change: 2006-01-25 20:40:30.000000000 +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();
EventDetails.getUpdatEvent();
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
```

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 is the annotation hierarchy in components/model/resources/mappings/annotations.ome.xml[16]. Hibernate supports this polymorphism, and will search all subclasses when a super class 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()[17].

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 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.

[16] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/resources/mappings/annotations.ome.xml
[17] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/src/ome/util/Utils.java

Limitations

- The filter methods override all standard checks such as IObject#isLoaded 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.

18.1.3 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.

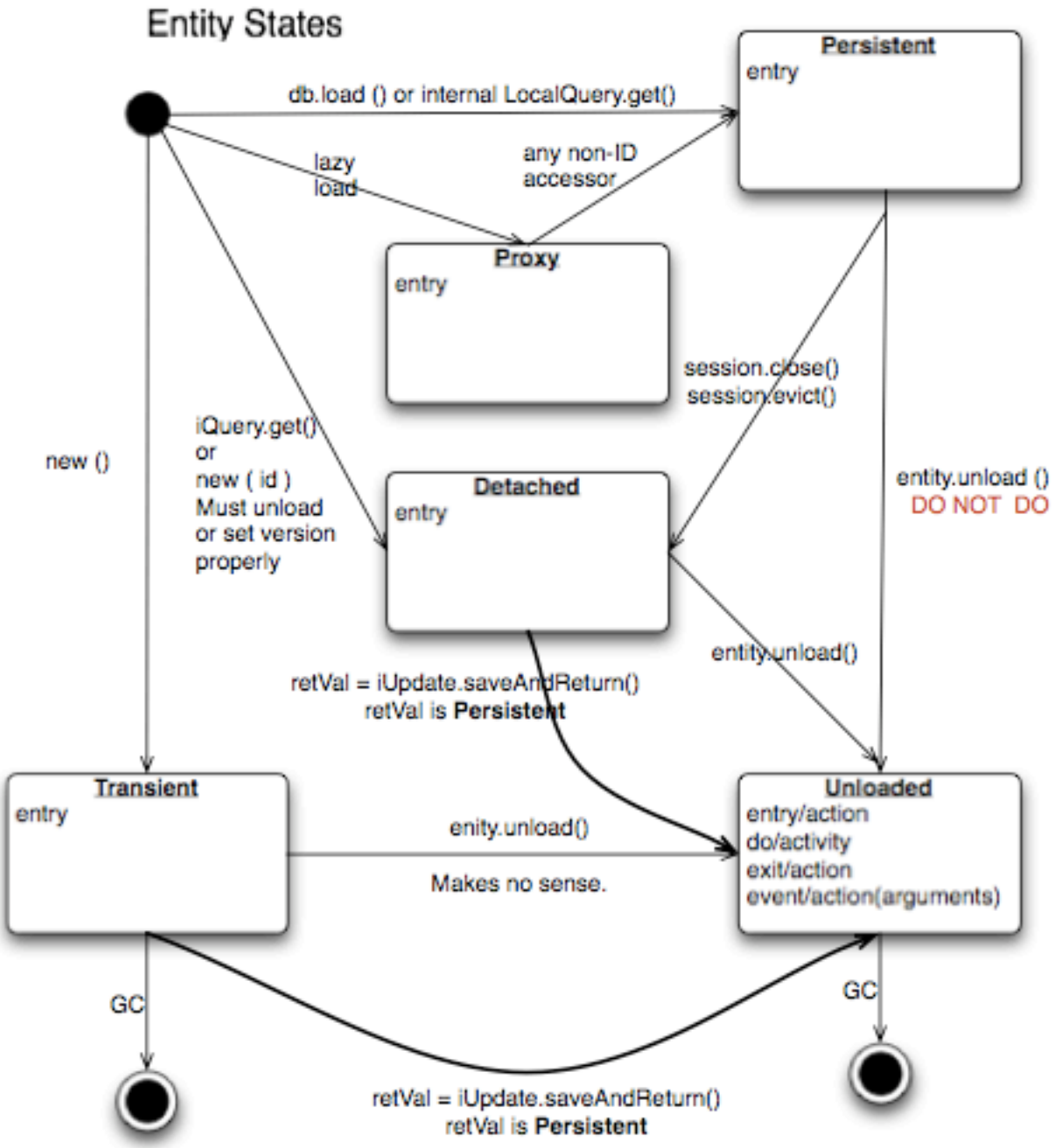
Class/id-based lookup is in fact so useful that it is possible to take an model object and call `obj.unload()` to have a "reference" – essentially a placeholder for a model object that contains only an id. Calls to any accessors other than get/setId will throw an exception. An object can be tested for loadedness with `obj.isLoaded()`.

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 through `LazyIniitalizationExceptions` 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
```



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 upates 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.

Note: There is one exception to this discussed below under *Links*. See that section or #1649[18] for more information.

18.1.4 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";
}
```

[18] http://trac.openmicroscopy.org.uk/ome/ticket/1649

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 logins 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 =
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
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, return oldProject, 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 `OptimisticLock-Exception`. See #1649[19] for more information.

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 only loads 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.isInitialized( ofile.getFormat() ) {
  ofile.setFormat( new Format( ofile.getFormat().getId(), false) );
}
```

[19] http://trac.openmicroscopy.org.uk/ome/ticket/1649

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[20] for the implementation.

Collections

More than just the nulling during serialization, collections pose several interesting problems.

For example, a collection may 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 they 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:

[20] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/tools/hibernate/ProxyCleanupFilter.java

```
someProject.linkDataset( someDataset );
updatedProject = update.saveAndReturnObject( someProject );

updatedProject.unlinkDataset( someDataset );
update.saveObject( updateProject ); // will no __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 ProjectDatasetLink (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.

18.1.5 Future topics

- Validation: Since the accessor methods themselves are largely logic-less, the work of validating the objects has been offset to validation objects and the Hibernate system. For each given object, a validation method can be specified which will check instance fields (TODO: the null-policy should be configurable based on whether or not the object is currently in a session). Validation is intended to verify the specification constraints which cannot (easily and/or quickly) be verified by the database.
- Versioning/Locking
- `ObjectFactory` for wrapping model objects from *OMERO.blitz*
- Links to external models
- Client cache
- Document collection methods
- Add info on the ILink interface to the section above.
- In addition to the extended functionality of the new object model, there are some changes to the actual structure, the specification, that are needed.
  - image_id ==> pixel_id where appropriate
  - plane_info
  - ACL (getting ownership in each table not MEX)
  - one table ; one class
  - cleaning up container relationships (project, category, screen, etc.)
  - replace ST definition ("ST is immutable") with locking meechanism
  - possibly versioning

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

18.2 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
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

18.2.1 Quality of transformations

	Targets									
Source	/2003-FC/	/2007-06/	/2008-02/	/2008-09/	/2009-09/	/2010-04/	/2010-06/	/2011-06/	/2012-06/	/2013-06/
/2003-FC/	—	excellent	excellent	excellent	excellent	excellent	excellent	excellent	excellent	excellent
/2007-06/	poor	—	excellent	excellent	excellent	excellent	excellent	excellent	excellent	excellent
/2008-02/	poor	poor	—	excellent	excellent	excellent	excellent	excellent	excellent	excellent
/2008-09/	poor	poor	poor	—	excellent	excellent	excellent	excellent	excellent	excellent
/2009-09/	poor	poor	poor	poor	—	excellent	excellent	excellent	excellent	excellent
/2010-04/	poor	poor	poor	fair	fair	—	excellent	excellent	excellent	excellent
/2010-06/	poor	poor	fair	fair	fair	fair	—	excellent	excellent	excellent
/2011-06/	poor	poor	fair	fair	fair	fair	good	—	excellent	excellent
/2012-06/	poor	poor	fair	fair	fair	fair	good	good	—	excellent
/2013-06/	poor	poor	fair	fair	fair	fair	good	good	good	—

18.2.2 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

18.2.3 Matrix of transformation paths

This shows the sequence of transformations used to convert one version of the schema to another version.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

18.3 Structured 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

	Targets								
Source	/2003-FC/	/2007-06/	/2008-02/	/2008-09/	/2009-09/	/2010-04/	/2010-06/	/2011-06/	/2012-06/
/2003-FC/	—	direct	via: /2007-06/	direct	via: /2008-09/	via: /2008-09/ & /2009-09/	via: /2008-09/ & /2009-09/ & /2010-04/	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ & /2011-06/
/2007-06/	via: 2008-09/ & /2009-09/ & /2010-04/ & /2010-06/	—	direct	direct	via: /2008-09/	via: /2008-09/ & /2009-09/	via: /2008-09/ & /2009-09/ & /2010-04/	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ & /2011-06/
/2008-02/	via: 2008-09/ & /2009-09/ & /2010-04/ & /2010-06/	via: /2003-FC/ & /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/	—	direct	via: /2008-09/	via: /2008-09/ & /2009-09/	via: /2008-09/ & /2009-09/ & /2010-04/	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/	via: /2008-09/ & /2009-09/ & /2010-04/ & /2010-06/ & /2011-06/
/2008-09/	via: /2009-09/ & /2010-04/ & /2010-06/	via: /2003-FC/ & /2009-09/ & /2010-04/ & /2010-06/	via: /2009-09/ & /2010-04/ & /2010-06/	—	direct	via: /2009-09/	via: /2009-09/ & /2010-04	via: /2009-09/ & /2010-04 & /2010-06/	via: /2009-09/ & /2010-04 & /2010-06/ & /2011-06/
/2009-09/	via: /2010-04/ & /2010-06/	via: /2003-FC/ & /2010-04/ & /2010-06/	via: /2010-04/ & /2010-06/	via: /2010-04/ & /2010/06/ & /2008-02/	—	direct	via: /2010-04/	via: /2010-04/ & /2010-06/	via: /2010-04/ & /2010-06/ & /2011-06/
/2010-04/	via: /2010-06/	via: /2003-FC/ & /2010-06/	via: /2010-06/	via: /2010/06/ & /2008-02/	via: /2010/06/ & /2008-02/ & /2008-09/	—	direct	via: /2010-06/	via: /2010-06/ & /2011-06/
/2010-06/	direct	via: /2003-FC/	direct	via: /2008-02/	via: /2008-02/ & /2008-09/	via: /2008-02/ & /2008-09/ & /2009-09/	—	direct	via: /2011-06
/2011-06/	via: /2010-06/	via: /2010-06/ & /2003-FC/	via: /2010-06/	via: /2010-06/ & /2008-02/	via: /2010-06/ & /2008-02/ & /2008-09/	via: /2010-06/ & /2008-02/ & /2008-09/ & /2009-09/	direct	—	direct
/2012-06/	via: /2011-06/ & /2010-06/	via: /2011-06/ & /2010-06/ & /2003-FC/	via: /2011-06/ & /2010-06/	via: /2011-06/ & /2010-06/ & /2008-02/	via: /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/	via: /2011-06/ & /2010-06/ & /2008-02/ & /2008-09/ & /2009-09/	via: /2011-06	direct	—

instances simultaneously to quickly annotated all entities in a view. Each annotation has a "name" which can be interpreted as a "namespace" by tools, which can filter out all unknown namespaces. Further, to prevent users from overwriting or editing important information, annotations are immutable, but editing can be simulated via copy and delete.

18.3.1 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`, `WellSample`, `Reagent`
- …

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
  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
    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[21] Section of the auto-generated schema documentation describing the structured annotations

18.3.2 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.

18.3.3 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);
```

18.3.4 Immutability

The actual content value of an annotation – the text, long, double, file value, etc – is immutable. Links to and from the annotation, however, can be modified.

Currently the namespace field of annotations is mutable. See #878[22] for discussion.

18.3.5 Examples

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

[22] http://trac.openmicroscopy.org.uk/ome/ticket/878

CHAPTER

NINETEEN

SEARCHING

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

19.1 OMERO search

Beginning with 3.0-Beta3, the OMERO server will use Lucene[1] to index all string and timestamp information in the database, as well as all `OriginalFiles` 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.

19.1.1 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.

[1] http://lucene.apache.org

Field	Comments
	Any unprefixed field searches the combination of all fields together i.e. a search for *cell AND name:myImage* gets translated to *combined_fields:cell AND name:myImage*.
<field name>	Each string, timestamp, or `Details` field of the entity also gets its own Field entry, like the `name` field above
details.owner.omeName	Login name of the owner of the object
details.owner.firstName	First name of the owner of the object
details.owner.lastName	Last name of the owner of the object
details.group.name	Group name of the owning group of the object
details.creationEvent.id	Id of the Event of this objects creation
details.creationEvent.time	When that Event took place
details.updateEvent.id	Id of the Event of this objects last modification
details.updateEvent.time	When that Event took place
details.permissions	Permissions in the form *rwrwrw* or *rw-*
tag	Contents from a `TagAnnotation`.
annotation	Contents from any annotations, including `TagAnnotation` and any `TextAnnotation` on another `TextAnnotation` (a.k.a. a *description*)
annotation.ns	Namespace (if present) for any annotations on an object
annotation.type	Short type name, e.g. `TextAnnotation` or `FileAnnotation` for any annotations on an object
file.name	For `FileAnnotations` and objects they are attached to, the name of the `OriginalFile`
file.format	For `FileAnnotations` and objects they are attached to, the format of the `OriginalFile`
file.path	For `FileAnnotations` and objects they are attached to, the path of the `OriginalFile`
file.sha1	For `FileAnnotations` and objects they are attached to, the sha1 of the `OriginalFile`
file.contents	For `FileAnnotations` and objects they are attached to as well as the `OriginalFile` itself, the file contents themselves if their Format is configured with the File parsers.
Internal,	
combined_fields	The default field prefix.
_hibernate_class	Used by Hibernate Search to record the entity type. The class value, e.g. ome.model.core.Image is also entered in combined_fields. Unimportant for the casual users.
id	The primary key of the entity. Unimportant for the casual user

19.1.2 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.

19.1.3 Indexing

Successful searching depends on understanding how the text is indexed. The default analyzer used is the FullTextAnalyzer[2].

```
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 and 2.
- searching for **"GFP H2B"** also returns 1 and 2.

[2] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/services/fulltext/FullTextAnalyzer.java

- searching for **GFP H2B** returns 1, 2, and 3, since the two terms are joined by an **OR**.

19.1.4 Information for developers and system administrators

Scheduling indexing

Indexing is not driven by the user, but happens automatically in the background. Automatic indexing occurs at the frequency defined in etc/omero.properties:

```
omero.search.cron=0,30 * * * * ?
omero.search.batch=100
```

which implies every thirty seconds of every hour, day, month, year, etc. During each iteration, 100 `EventLogs` will be loaded from the database and processed. Upon successful completion, the persistent count in the `configuration` table, will be incremented.

```
omero3=# select value from configuration where name = 'PersistentEventLogLoader.current_id';
 value
-------
 30983
(1 row)
```

If you have more than one `PersistentEventLogLoader.*` value in your database, then 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.currend_id
```

became

```
PersistentEventLogLoader.v2.current_id
```

in r2460.

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 `oma.api.Search` interface (below).

If you need to re-index your database, stop your server, and:

- (Optionally) Delete the `/OMERO/FullText` directory
- Delete or set to 0 the entry from the `configuration` table: `update configuration set value = 0 where name like 'PersistentEventLogLoader%';`
- If it is necessary to force re-indexing, use:

```
cd $OMERO_PREFIX
CLASSPATH=etc:`find lib/server | xargs | sed 's/ /:/g'`
java -Dlog4j.configuration=log4j-cli.properties -Xmx512M ome.services.fulltext.Main full
```

or alternatively for particular types, ...

```
java -Dlog4j.configuration=log4j-cli.properties -Xmx512M ome.services.fulltext.Main reindex ome.model.cc
```

This functionality is still being tested, but will be made more available in future versions.

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 silently be ignored (I).

Query Method –> Parameters	byFullText/SomeMustNone	byGroupForTags/byTagsForGroup	byAnnotatedWith
annotated between	S	S	S
annotated by	S	S	S
annotated with	S	I	I
created between	S	S	S
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	[7]	I	[8]
Other			
setProjections [9]	X	X	X
current*Metdata [10]	X	X	X
setProjections [3]	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);
```

[3] Any fetchAnnotation() argument to byFullText() or related queries, returns **all** annotations.
[4] byAnnotatedWith() does not accept a fetchAnnotation() argument of `Annotation.class`.
[5] setProjects may need to be removed if Lucene cannot handle OMERO's security requirements.
[6] Not yet implemented.
[7] Any fetchAnnotation() argument to byFullText() or related queries, returns **all** annotations.
[8] byAnnotatedWith() does not accept a fetchAnnotation() argument of `Annotation.class`.
[9] setProjects may need to be removed if Lucene cannot handle OMERO's security requirements.
[10] Not yet implemented.

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:

```
ACELL OR BCELL OR CCELL
```

If there are more than "omero.search.maxclause" terms in the expansion (default is 4096), 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[11]) 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:

Structured annotations, *Search bridges*, *File parsers*, Query Parser Syntax[12],

Luke[13] a Java application which you can download and point at your `/OMERO/FullText` directory to get a better feeling for Lucene queries.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

19.2 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.

19.2.1 Configuration

Currently, configuration takes places solely in service-ome.api.Search.xml[14]. Eventually, it should be able to replace file parsers at configuration or even runtime.

19.2.2 Available parsers

File type	Parser
application/pdf	http://pdfbox.apache.org
text/xml	(internal)
text/plain	(internal)
text/csv	(internal)

The base class for File parsers are FileParser.java[15]

[11] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/services/fulltext/FullTextBridge.java
[12] http://lucene.apache.org/core/3_6_0/queryparsersyntax.html
[14] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/service-ome.api.Search.xml
[15] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/services/fulltext/FileParser.java

See also:

OMERO search

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

19.3 Search bridges

A "bridge" is the mapping between your metadata and how it is stored in the Lucene[16] 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 components/server/src/ome/services/fulltext/BridgeHelper.java[17]

19.3.1 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);

 }
```

19.3.2 Configuration

Custom bridges are configured in etc/omero.properties 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 components/server/src/ome/services/fulltext/FullTextBridge.java[18].

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*.

19.3.3 Available bridges

See components/server/src/ome/services/fulltext/bridges[19] for a list of provided (example) bridges.

19.3.4 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

[16] http://lucene.apache.org
[17] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/services/fulltext/BridgeHelper.java
[18] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/services/fulltext/FullTextBridge.java
[19] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/server/src/ome/services/fulltext/bridges

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);
                 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[20] and #1102[21]

See also:

OMERO search

[20] http://trac.openmicroscopy.org.uk/ome/ticket/955
[21] http://trac.openmicroscopy.org.uk/ome/ticket/1102

CHAPTER

TWENTY

AUTHENTICATION AND SECURITY

20.1 Password Provider

A *Password Provider* is an implementation of the Java interface ome.security.auth.PasswordProvider[1]. Several implementations exist currently:

- ome.security.auth.JdbcPasswordProvider[2] is the most common provider, and uses the "password" table for storing passwords hashed using MD5 and salt per user.
- ome.security.auth.FilePasswordProvider[3] 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[4] 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 "chainedPasswordProvider" (ome.security.auth.PasswordProviders[5]) is configured for use by default in etc/omero.properties[6] under `omero.security.password_provider`. It first checks with the `LdapPasswordProvider` and then falls back to the `JdbcPasswordProvider`.

To write your own provider, you can either subclass from ome.security.auth.ConfigurablePasswordProvider[7] as the providers above do, or write your own implementation from scratch. You will need to define your object in a Spring XML file matching the pattern `ome/services/db-*.xml`. See *Extending OMERO* more for information.

20.1.1 Things to keep in mind

- All the existing implementations take care to publish a LoginAttemptMessage[8] 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

[1] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/PasswordProvider.java
[2] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/JdbcPasswordProvider.java
[3] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/FilePasswordProvider.java
[4] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/LdapPasswordProvider.java
[5] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/PasswordProviders.java
[6] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/omero.properties
[7] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/ConfigurablePasswordProvider.java
[8] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/services/messages/LoginAttemptMessage.java

20.2 LoginAttemptListener

All the *Password Provider* implementations provided by default publish a "LoginAttemptMessage[9]" 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[10] which throttles logins after a given number of failed attempts. Configuration for this listener is available in etc/omero.properties[11]:

```
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[12].

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

20.3 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.

20.3.1 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 non-null `dn` in the `password` 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*.
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:
   (a) The `user_mapping` property is of the form: `omeName=<ldap attribute>;firstName=<ldapAttribute>;…`
   (b) 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)`.
   (c) 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`.
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.

20.3.2 NewUserGroupBean.java

The interface described for the ":bean:" `new_user_group` prefix, is ome.security.auth.NewUserGroupBean[13]. It defines a single method: `groups(…, AttributeSet set)` which returns a list of `ExperimenterGroup` ids (`List<Long>`) which the user should be added to.

[9] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/services/messages/LoginAttemptMessage.java
[10] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/LoginAttemptListener.java
[11] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/omero.properties
[12] http://trac.openmicroscopy.org.uk/ome/ticket/3139
[13] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/NewUserGroupBean.java

Other prefix handlers also implement the interface as examples. In the same package are:

- **`:attribute: -`** AttributeNewUserGroupBean.java[14]
- **`:ou: -`** OrgUnitNewUserGroupBean[15]
- **`:query: -`** QueryNewUserGroupBean[16]

See also:

OMERO.server installation Instructions for installing OMERO.server on UNIX and UNIX-like platforms

OMERO.server installation Instructions for installing OMERO.server on Windows platforms

Server security and firewalls

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

20.4 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".

20.4.1 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)`

20.4.2 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.

---

[14] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/AttributeNewUserGroupBean.java
[15] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/OrgUnitNewUserGroupBean.java
[16] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/auth/QueryNewUserGroupBean.java

20.4.3 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.

20.4.4 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

20.5 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*
- *Permissions overview*
- *OMERO permissions history, querying and usage*

20.5.1 Concepts

Several concepts and/or components from our and other code bases play a role in OMERO security:

Hibernate Listeners and Events[17] 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[18] 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.

20.5.2 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.properties[19] contains login and connection information for the database and OMERO.

local.properties.example[20] contains the default root password. This can be overridden with your own `etc/local.properties` file.

hibernate.properties[21] 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[22] 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.

mapping.vm[23] specifies the default permissions that all objects will have after construction, as well as attaches the security filter to all classes and collections.

data.vm[24] used by DSLTask to generate psql-footer.sql which is used to bootstrap the database security system (root et al).

common/build.xml[25] contains an ant target (adduser) which will create a user and empty password from the command line. This target can also be called from the top-level (java omero adduser).

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[26] is used.

IAdmin.java[27] main interface for administering accounts and privileges. See *OMERO admin interface* for more.

ITypes.java[28] 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[29] 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[30] contains all the fields necessary to perform access control, such as owner, group, and permissions.

Permissions.java[31] representation of rights and roles. For more information, see *Permissions overview*.

Token.java[32] 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[33] 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[34] the exception thrown by the *OMERO security system* at the first hint of misdoings.

Principal.java[35] an Omero-speciific 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[36]

JBoss-only

ServiceFactory.java[37] Login.java[38] Server.java[39]

[26] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/resources/ome/config.xml
[36] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/model/resources/mappings/meta.ome.xml
[37] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/system/ServiceFactory.java
[38] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/system/Login.java
[39] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/system/Server.java

Server side

AdminImpl.java[40] CurrentDetails.java[41] SecureAction.java[42] SecuritySystem.java[43] BasicSecuritySystem.java[44] ACLEventListener.java[45] EventHandler.java[46] MergeEventListener.java[47] OmeroInterceptor.java[48] SessionHandler.java[49] SecurityFilter.java[50] EventLogListener.java[51] EventListenersFactoryBean.java[52] LocalAdmin.java[53] hibernate.xml[54] sec-system.xml[55] services.xml[56]

20.5.3 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 from /etc 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.

[40] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/logic/AdminImpl.java
[41] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/basic/CurrentDetails.java
[42] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/SecureAction.java
[43] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/SecuritySystem.java
[44] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/basic/BasicSecuritySystem.java
[45] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/ACLEventListener.java
[46] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/basic/EventHandler.java
[47] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/basic/MergeEventListener.java
[48] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/basic/OmeroInterceptor.java
[49] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/tools/hibernate/SessionHandler.java
[50] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/SecurityFilter.java
[51] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/basic/EventLogListener.java
[52] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/basic/EventListenersFactoryBean.java
[53] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/api/local/LocalAdmin.java
[54] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/hibernate.xml
[55] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/sec-system.xml
[56] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/services.xml

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 instatiates 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[57]) 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 apears in a query.

[57] http://trac.openmicroscopy.org.uk/ome/ticket/223

```
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[58].

The *OMERO security system* and its current only implementation BasicSecuritySystem? are somewhat inert and expect well-defined and trusted (see #235[59]) methods to invoke callbacks during the proper Hibernate phase.

20.5.4 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[60] instance. This is then passed to the server on each invocation of a proxy obtained from JNDI.

20.5.5 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[61].

Basically it amounts to this:

```
Principal p = new Principal( omeroUserName, omeroGroupName, omeroEventTypeValue );
securitySystem.login( p );
```

This must be run otherwise the EventHandler[62] will throw a security exception.

Note: The code above is being run in a secure context (i.e. you are root). Please be careful.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

20.6 OMERO permissions history, querying and usage

20.6.1 Introduction

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.

[58] http://trac.openmicroscopy.org.uk/ome/ticket/234
[59] http://trac.openmicroscopy.org.uk/ome/ticket/235
[60] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/system/Principal.java
[61] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/SecuritySystem.java
[62] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/basic/EventHandler.java

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.

See also:

OMERO 4.2.0 Server Permissions[63] Database upgrade from 4.1 to 4.2[64] *Deleting in OMERO*

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.

20.6.2 Working with the OMERO 4.4.x permissions system

Example environment

- OMERO 4.4.8 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

[63] http://www.openmicroscopy.org/site/support/previous/omero420/server/permissions
[64] http://www.openmicroscopy.org/site/support/previous/omero420/server/db-upgrade-41-to-42

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
    sessionUuid = 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
        [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')

# 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.
     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.
# 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' % \
    [(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()[65]
- canDelete()[66]
- canEdit[67]
- canLink[68]

20.6.3 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 4.4.x 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.

[65] http://ci.openmicroscopy.org/job/OMERO-stable/javadoc/slice2html/omero/model/Permissions.html#canAnnotate
[66] http://ci.openmicroscopy.org/job/OMERO-stable/javadoc/slice2html/omero/model/Permissions.html#canDelete
[67] http://ci.openmicroscopy.org/job/OMERO-stable/javadoc/slice2html/omero/model/Permissions.html#canEdit
[68] http://ci.openmicroscopy.org/job/OMERO-stable/javadoc/slice2html/omero/model/Permissions.html#canLink

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.

CHAPTER

TWENTYONE

EXTENDING OMERO SERVER

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.1 OMERO.server overview

21.1.1 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[1] is examined for a valid *session* which was created via ISession.createSession(String username, String password)[2].
- 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[3] for how `root` and the intial 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[4] 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[5] or ome.services[6]) will be either read-only (IQuery[7]-based) or a read-write (IUpdate[8]-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.

[1] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/system/Principal.java
[2] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/ISession.java
[3] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/dsl/resources/ome/dsl/psql-footer.vm
[4] http://www.oracle.com/technetwork/java/javaee/ejb/index.html
[5] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/server/src/ome/logic
[6] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/server/src/ome/services
[7] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IQuery.java
[8] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IUpdate.java

- After execution, the OMERO runtime is reset, the method call is logged, and either the successful results are returned or an *exception* is thrown.

21.1.2 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[9] 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.

21.1.3 Server design

The server logic resides in the components/server[10] component.

Topics

- *Exception handling*
- *OME-Remote Objects*
- *Server security and firewalls*

See also:

OMERO.grid

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.2 Extending OMERO

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 ome-devel mailing list[a] with any such suggestions.

[a] http://lists.openmicroscopy.org.uk/mailman/listinfo/ome-devel/

[9] http://www.hibernate.org
[10] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/server

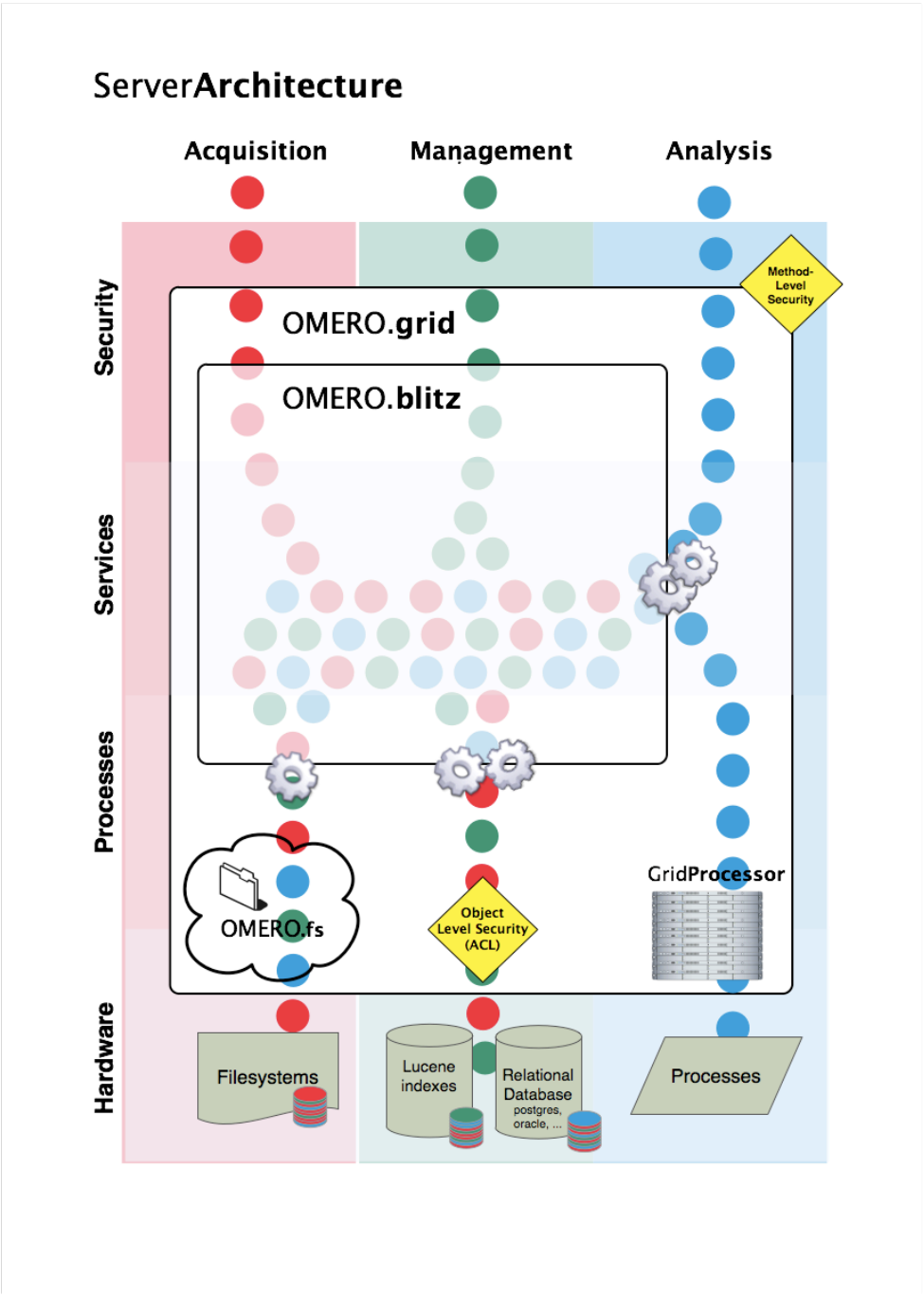


Figure 21.1: Server Architecture

21.2.1 Other starting points

Clients

To write you own clients or scripts against the OMERO API, see the *Working with OMERO* page. Though writing your own client is a form of extending OMERO, the topics that follow are for extending the server and do not cover clients. For information specific to OMERO.insight, please see the client documentation:

- *How to build an agent*
- *How to build an agent's view*
- *Retrieve data from server*

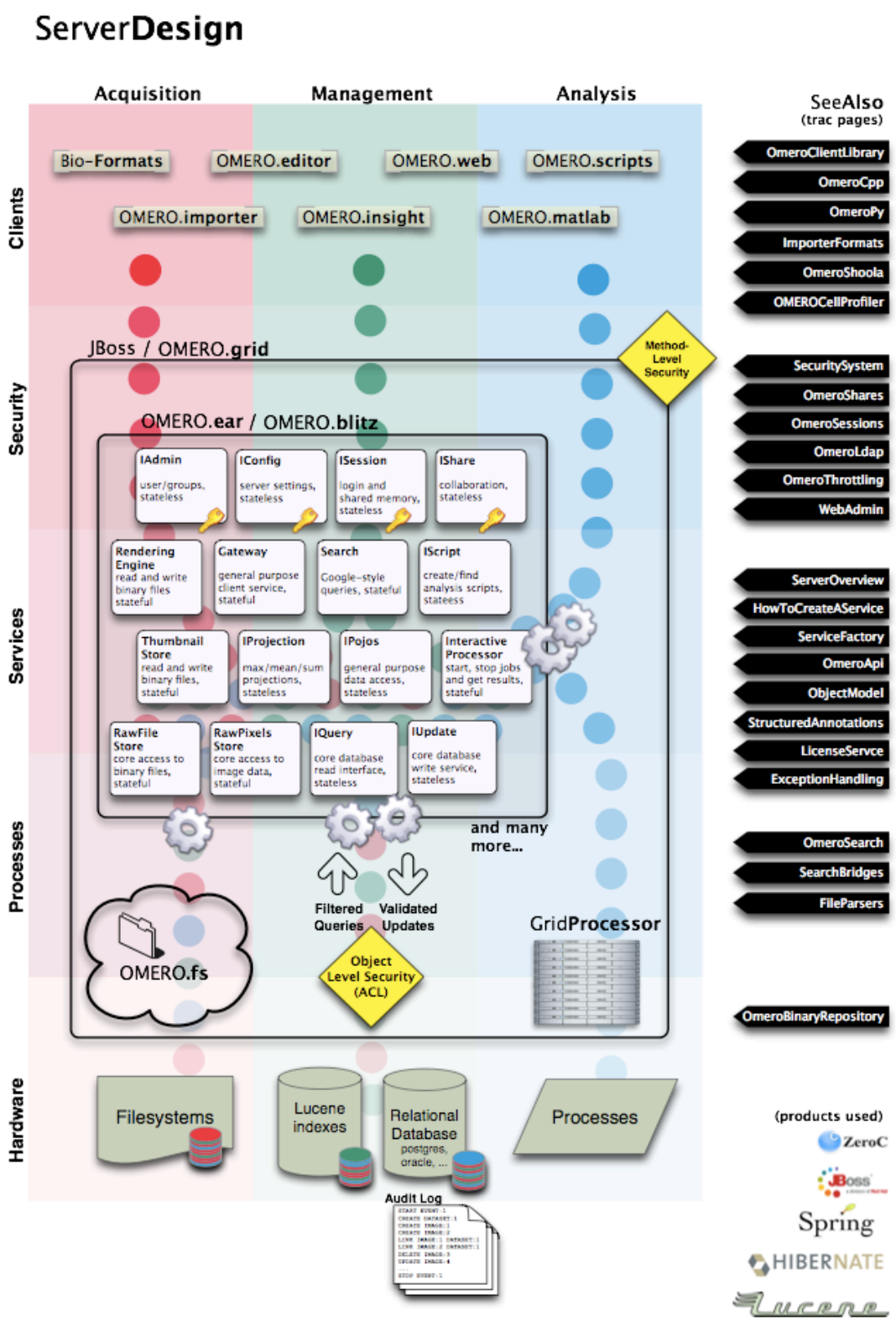


Figure 21.2: Server Design

List of 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

- *OMERO Command Line Interface* - 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

21.2.2 Main topics

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 *Structured 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 components/model/resources/mappings[11] 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.

[11] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/model/resources/mappings

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 components/common/src/ome/api[12] 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 components/blitz/resources/omero[13].

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 `blitz/resources/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;
}
```

[12] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/common/src/ome/api
[13] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/blitz/resources/omero

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[14].

Java configuration

Configuration in the Java servers takes place via Spring[15]. 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 components/blitz/resources/ome/services[16] and components/server/resources/ome/services[17] 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)

[14] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/blitz/resources/ome/services/blitz-servantDefinitions.xml#L36
[15] http://spring.io
[16] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/blitz/resources/ome/services
[17] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/server/resources/ome/services

Future topics

Information on:

- implementation, configuration, and deploy in other *OMERO.blitz* languages
- Subclassing from existing servant implementation
- Using AMD to reduce server contention

will be provided in the future or upon request.

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>
  <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 -1
MyLoginListener.class
MyLoginListener.java
```

```
lib
...
~/example $ jar cvf lib/server/extensions.jar MyLoginListener.class ome/services/blitz-myLoginListener.
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[18] is started to host *OMERO.web framework*. Better integration is possible however, if your server also uses the Ice[19] 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:

```
bin/omero admin start
```

the application descriptor used is etc/grid/default.xml[20]. 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"/>
  </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:

[18] http://wiki.nginx.org/Main
[19] http://www.zeroc.com
[20] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/default.xml

[ome-devel] model description driven code generation[21]

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.3 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*.

21.3.1 Metadata

Metadata stored in an object-relational database is mapped into the OMERO *OME-Remote Objects* via Hibernate[22]. Hibernate Query Language (HQL) calls can be made against the server and have all ownership information automatically taken into account.

21.3.2 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.4 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: components/tools/OmeroPy/src/omero/processor.py[23]
- Documentation: *OMERO.grid*

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.5 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

[21] http://lists.openmicroscopy.org.uk/pipermail/ome-devel/2009-July/001332.html
[22] http://www.hibernate.org
[23] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/tools/OmeroPy/src/omero/processor.py

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[24].

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.6 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.

21.6.1 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 down-time. Currently, *OMERO sessions* are sticky to a cluster node and so it is not possible shutdown a node at any time. All new sessions can be redirected to the server which is to be left turned on however, then when all active sessions have completed, the chosen server can be shutdown.

21.6.2 Throughput

The other main reason to have other servers running is to service more user sessions simultaneously. Out of the box, each *OMERO.blitz* process is configured for 400MB of memory. 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, and so to further extend throughput, you will need to parallelize these.

21.6.3 Installation

If you are using the default *OMERO.grid* application descriptor[25] quickly enabling clustering is as simple as executing:

```
bin/omero config set omero.cluster.redirector configRedirector
bin/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

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.7 Collection counts

The IContainer[26] interface has always provided a method for returning the count of some collection types via `getDetails().getCounts()`. Previous to 3.0-Beta3, the counting process was fairly time intensive, and has been removed.

[24] http://www.zeroc.com
[25] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/etc/grid/default.xml
[26] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IContainer.java

In its place, the 3.0-Beta3 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.

21.7.1 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.

21.7.2 Restrictions

Currently a Hibernate bug (waiting to be filed) prevents retrieving the counts on any other than the top-level object ("select this").

21.7.3 Instructions

Starting with `OMERO3A__3`, the views.sql script is automatically executed when initializing your database. If you have an older database, upgrade it to the latest version, and apply the `views.sql` manually.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.8 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*. If you would indeed like to create a core service, please contact the ome-devel mailing[a] list

[a] http://www.openmicroscopy.org/site/community/mailing-lists

To fulfill #306[27], 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[28], an implementation[29] of that interface, a Spring configuration file[30], as well as modify the server configuration[31] and the central service factory[32] (These last two points stand to change with #314[33]).

[27] http://trac.openmicroscopy.org.uk/ome/ticket/306
[28] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/api/IConfig.java
[29] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/logic/ConfigImpl.java
[30] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/service-ome.api.IConfig.xml
[31] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/
[32] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/system/ServiceFactory.java
[33] http://trac.openmicroscopy.org.uk/ome/ticket/314

Note: With the creation of *OMERO.blitz*, there are several other locations which need to be modified. These are also listed below.

21.8.1 Files to create

components/common/src/ome/api/IConfig.java[34] 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''**.

components/server/src/ome/logic/ConfigImpl.java[35] the implementation which will usually subclass `AbstractLevel{1,2}Service` or `AbstractBean` (See more below on **super-classes**) This is 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.

components/server/resources/ome/services/service-ome.api.IConfig.xml[36] a Spring[37] 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[38] file (see below).

components/blitz/src/ome/services/blitz/impl/ConfigI.java[39] a Ice[40] "servant" implementation which can use on of several methods for delegating to the `ome.api.IConfig` interface, but all of which support *throttling*.

21.8.2 Files to edit (not strictly necessary, see #314)

components/common/src/ome/system/ServiceFactory.java[41] our central API factory, needs an additional method for looking up the new interface (**get<interface name>Service()**)

components/server/resources/ome/services/[42] **server** **Spring**[43] configurations, which makes the use of JNDI and JAAS significantly simpler.

components/blitz/resources/omero/API.ice[44] **(blitz)** a ZeroC[45] slice definition file, which provides cross-language mappings. Add the same service method to `ServiceFactoryI` as to `ServiceFactory.java`.

components/blitz/resources/ome/services/blitz-servantDefinitions.xml[46] **(blitz)** a Spring[47] configuration, which defines a mapping between Ice servants and Java services.

components/blitz/resources/omero/Constants.ice[48] **(blitz)** a ZeroC[49] slice definition file, which provides constants needed for looking up services, etc.

components/blitz/src/ome/services/blitz/impl/ServiceFactoryI.java[50] **(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.

21.8.3 Files involved

components/server/resources/beanRefContext.xml[51]

components/blitz/resources/beanRefContext.xml[52] Singleton definitions[53] 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[54], 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.

[37] http://spring.io
[38] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/beanRefContext.xml
[40] http://www.zeroc.com
[45] http://www.zeroc.com
[47] http://spring.io
[49] http://www.zeroc.com
[51] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/beanRefContext.xml
[53] http://docs.spring.io/spring/docs/2.0.x/reference/beans.html#beans-factory-scopes-singleton
[54] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/beanRefContext.xml

21.8.4 And do not forget the tests

components/server/test/ome/server/itests/ConfigTest.java[55] tests only the implementation without a container.

blitz: Currently, testing blitz is outside the scope of this document.

21.8.5 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[56]. 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[57] for an example.

Stateful services

Currently all stateful services are in their own component (components/rendering[58] and components/romio[59]) but their interface will still need to be under components/common[60] for them to be accessible to clients. To be done.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.9 OMERO sessions

Beginning with OMERO-3.0-Beta3, the OMERO server has unified the handling of login sessions among both the JBoss and the *OMERO.blitz* servers. Previously JBoss logins were handled via the standard JAAS[61] mechanisms, using a modified `DatabaseLoginModule`. This proved problematic for several reasons:

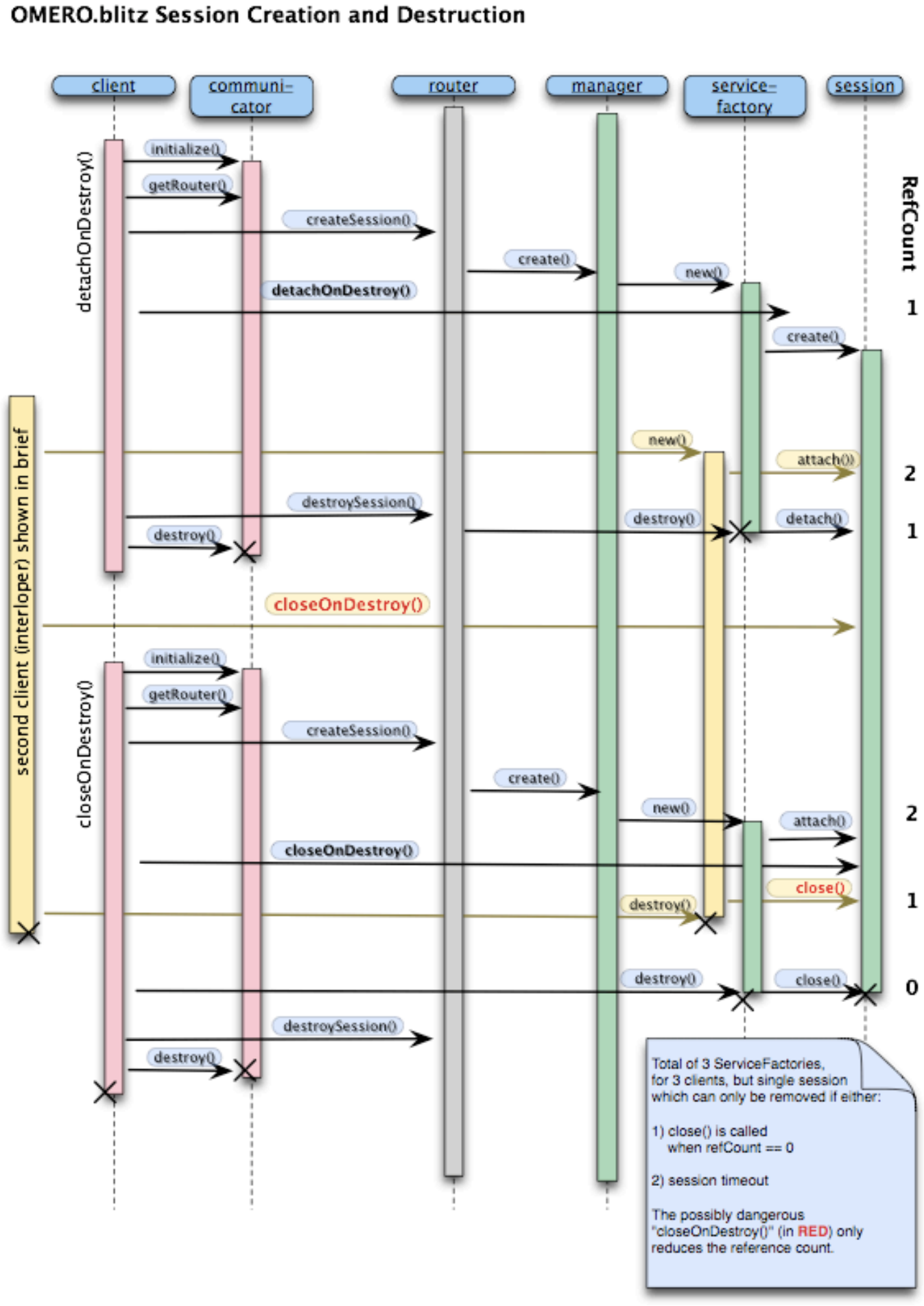
1. Exceptions thrown during login could not easily be caught or specified.
2. Passwords were sent in the clear on every invocation.
3. Sql queries on every invocation caused significant overhead.

Blitz did not suffer from these problems, but the login functionality was largely outside of the core server code and sessions were more volatile: a loss of an Ice connection caused all resources to be lost. With *OMERO sessions*, both login systems have been brought together and simplified.

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.

[56] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/server/src/ome/api/local
[57] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/logic/UpdateImpl.java
[58] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/rendering
[59] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/romio
[60] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/common
[61] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html


OMERO.blitz Session Creation and Destruction
client
communi-cator
router
manager
service-factory
session
RefCount
detachOnDestroy()
initialize()
getRouter()
createSession()
create()
new()
detachOnDestroy()
create()
1
new()
attach()
2
destroySession()
destroy()
destroy()
detach()
1
closeOnDestroy()
second client (interloper) shown in brief
closeOnDestroy()
initialize()
getRouter()
createSession()
create()
new()
attach()
2
closeOnDestroy()
close()
destroy()
1
destroy()
close()
0
destroySession()
destroy()
Total of 3 ServiceFactories, for 3 clients, but single session which can only be removed if either:
1) close() is called when refCount == 0
2) session timeout
The possibly dangerous "closeOnDestroy()" (in RED) only reduces the reference count.

- Sessions prevent sending passwords in plain text or any other form. After that, all session interactions take place via a shared secret key.

21.9.1 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 princpal, 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:

```
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);
```

21.9.2 Existing sessions

In *OMERO.blitz*, once the connection to a `ServiceFactoryPrx` (a Glacier2.Session subclass) was lost, it was not possible to reconnect to any of the services created using that connection. Now 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
```

21.9.3 Backwards compatibility

In the short-term, there is no need for any change to client code to make use of the new sessions.

`ome.system.ServiceFactory` has been modified to automatically acquire a session before the first service call is made. Eventually, clients will want to make use of the session API and catch session exceptions to have a finer control of the client lifecycle.

Similarly, no changes are needed in *OMERO.blitz* client code since Glacier2 sessions now delegate to *OMERO sessions*. Clients can access the ISession service when necessary. Exceptions thrown are still Ice-based.

See also:

Server security and firewalls

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.10 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[62], advisors are declared in the bean definition files (under components/server/resources/ome/services[63], services.xml[64], hibernate.xml[65], 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[66], 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.

21.10.1 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:

[62] http://spring.io
[63] https://github.com/openmicroscopy/openmicroscopy/tree/v.4.4.12/components/server/resources/ome/services
[64] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/services.xml
[65] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/hibernate.xml
[66] http://eclipse.org/aspectj/

```
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[67] Chapter of the Spring documentation

AOP Alliance [68] Joint project defining interfaces for various AOP implementations

AspectJ[69] The arguable leader in Java/AOP development. Not used in Omero, but a good starting point.

Aspect-oriented programming[70] Wikipedia page on AOP

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.11 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* (components/common/src/ome/system/OmeroContext.java[71]). Based on the Spring[72] configuration backing the context, it can be one of `client`, `internal`, or `managed`. The use of a ServiceFactory simplifies this usage for the client.

21.11.1 Hibernate sessions

A `Hibernate Session` comprises a Unit-of-Work[73] 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.

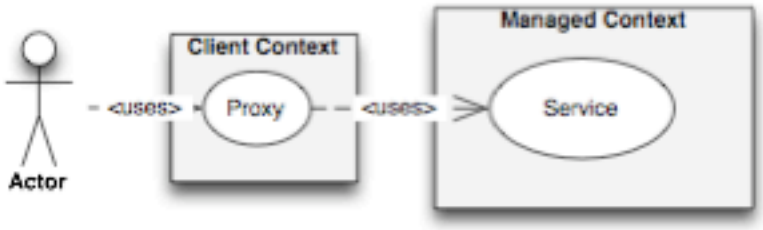
A session can span multiple calls by being disconnected from the underlying database transaction, and then reconnected to a new transaction on the next call (see components/server/src/ome/tools/hibernate/SessionHandler.java[74] for the implementation).

For information about Events see *OMERO events and provenance*.

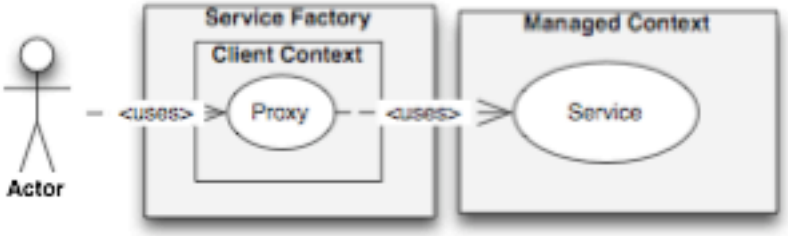
This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

[71] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/src/ome/system/OmeroContext.java
[72] http://spring.io
[73] http://www.martinfowler.com/eaaCatalog/unitOfWork.html
[74] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/tools/hibernate/SessionHandler.java

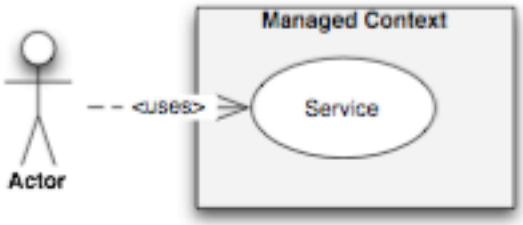
Omero Context Usage



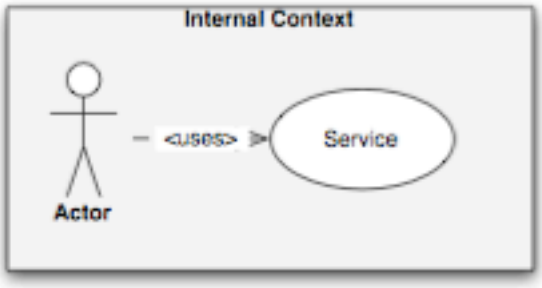
Client context: user accesses a proxy acquired from a context instantiated with `OmeroContext.getClientContext()` All objects are serialized and a transaction starts at the barrier to the Managed Context (see below).



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.

21.12 OMERO events and provenance

21.12.1 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 components/server/src/ome/security/basic/EventHandler.java[75] is responsible for creating new events.

21.12.2 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`).

21.12.3 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[76]

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.13 Properties

As of milestone OMERO-Beta4[77] (#800[78]), client usage of these properties has significantly changed. Please see the *sysadmin documentation* for how to configure your server installation.

Under the `etc/` directory in both the source and the binary distributions, several files are provided which help to configure OMERO.server:

```
etc/omero.properties            - Our central configuration file with all defaults.
etc/hibernate.properties        - Required by Hibernate since some properties are only configurable
                                  via a classpath:hibernate.properties file.
etc/log4j.xml                   - Logging configuration
etc/local.properties.example    - The properties that you will most likely want to change.
                                  This file can be copied to etc/local.properties to being, or
                                  alternatively you can run "java omero setup"
                                  (Name will change to "…default")
etc/local.properties            - Local overrides for other properties (used by build only)
```

During the build, these files get stored in the `blitz.jar` and are read-only. 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[79])

[75] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/src/ome/security/basic/EventHandler.java
[76] http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/events.html
[77] http://trac.openmicroscopy.org.uk/ome/milestone/OMERO-Beta4
[78] http://trac.openmicroscopy.org.uk/ome/ticket/800
[79] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/common/resources/ome/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:

- components/server/resources/ome/services/services.xml[80]

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>
```

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.14 Queries

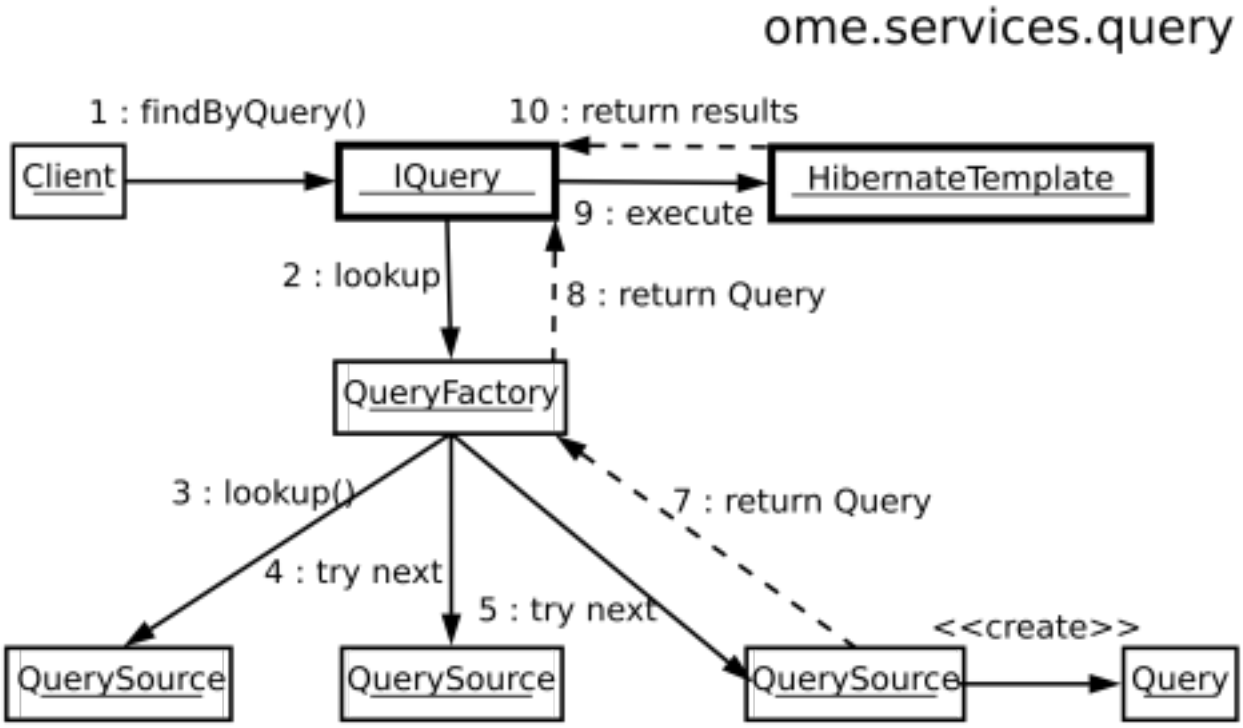


Figure 21.3: omero.services.query

21.14.1 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`.

[80] https://github.com/openmicroscopy/openmicroscopy/blob/v.4.4.12/components/server/resources/ome/services/services.xml

21.14.2 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","<myNamHere>");
```

Positional parameters of the form

```
select e from Experimenter where omeName = ?
```

are not supported.

21.14.3 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.

21.14.4 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.15 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 my any number of configurable **slots**. Each site can configure the number and type of slots based on which **throttling strategy** has been chosen.

21.15.1 Planning

Planned for milestone:3.0-Beta4, the infrastructure for throttling was committed to milestone 3.0-Beta3.1[81] 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.

21.15.2 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

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.16 OMERO rendering engine

21.16.1 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.

[81] http://trac.openmicroscopy.org.uk/ome/milestone/3.0-Beta3.1

21.16.2 Server-port

The rendering engine has been ported to also now sit on the server-side, though equally usable from any Java setting.

21.16.3 Optimizations

Here we have a listing of the various rendering engine optimizations that have taken place over time:

- Packed Integers (#449[82])
- Region Based Rendering (#450[83])
- Removal of RGB Rendering Model (#452[84])

21.16.4 Compression

With r1744 and r1748, the rendering engine now supports compression. (#6[85])

21.16.5 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.

This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.17 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.

21.17.1 Concurrent invocations

The bottlenecks for concurrent invocations are:

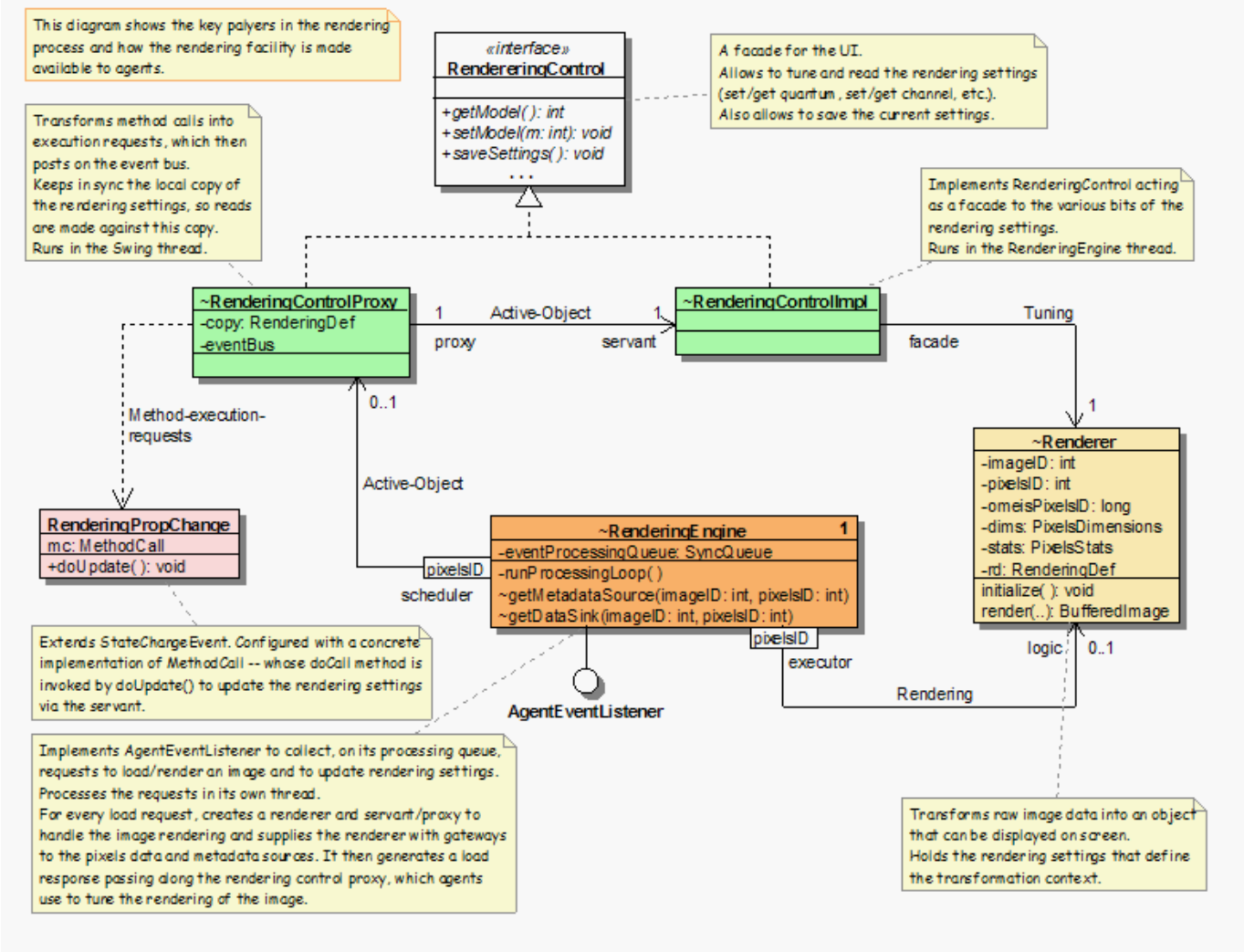
- database connections
- server threads
- the router

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.

With the default `max_connection` set to 64, it is possible to execute 500 queries simultaneously without database exceptions. Instead, one receives server exceptions.

[82] http://trac.openmicroscopy.org.uk/ome/ticket/449
[83] http://trac.openmicroscopy.org.uk/ome/ticket/450
[84] http://trac.openmicroscopy.org.uk/ome/ticket/452
[85] http://trac.openmicroscopy.org.uk/ome/ticket/6



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.

21.17.2 Total throughput

The bottlenecks for throughput are:

- maximum message size
- server memory
- IO
- network

See also:

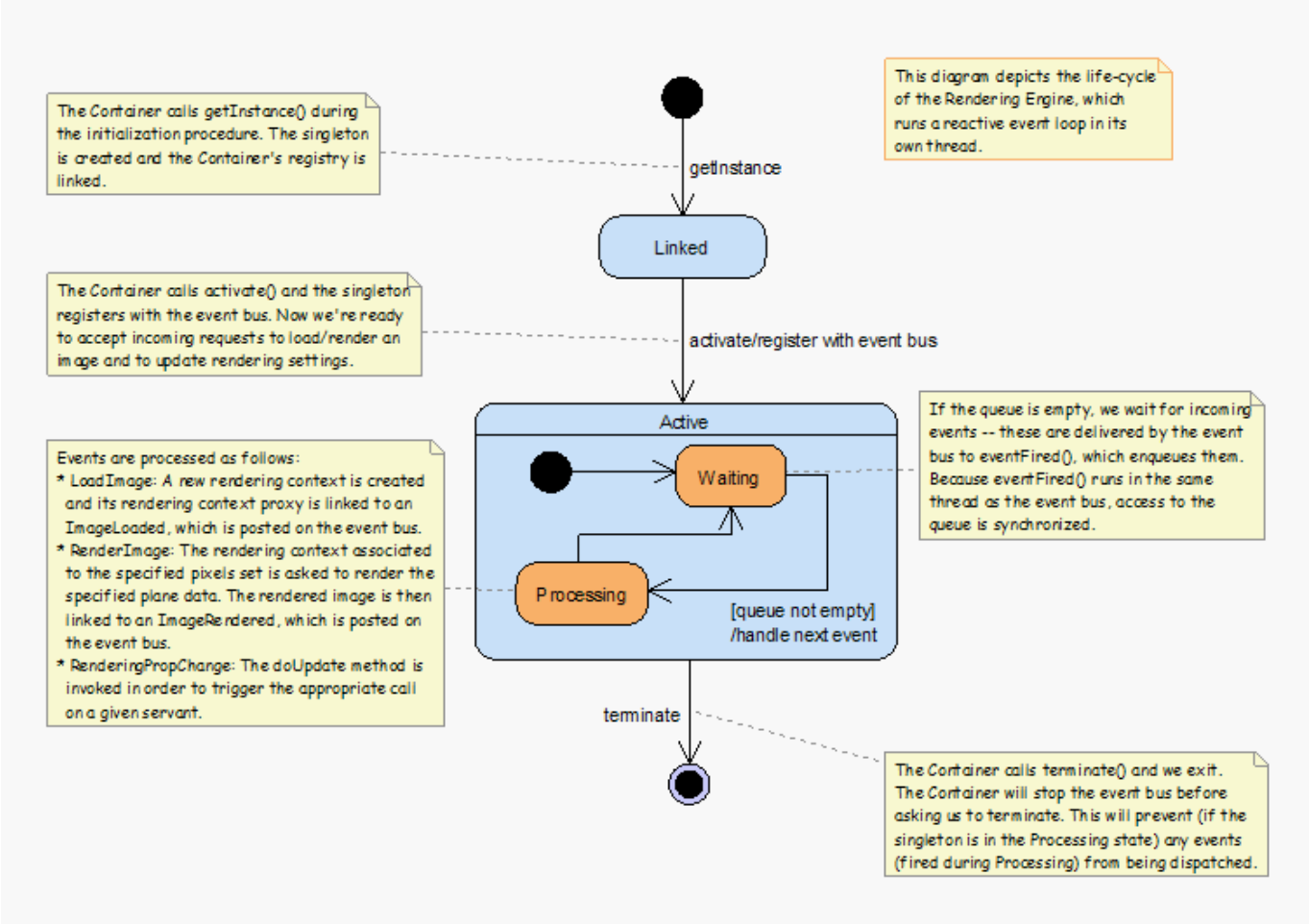
OMERO.server and PostgreSQL Instructions about OMERO.server and PostgreSQL under UNIX & UNIX-like platforms.

OMERO.server and PostgreSQL Instructions about OMERO.server and PostgreSQL under Windows platforms.

OMERO.grid

#906[86]

[86] http://trac.openmicroscopy.org.uk/ome/ticket/906



This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

21.18 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.

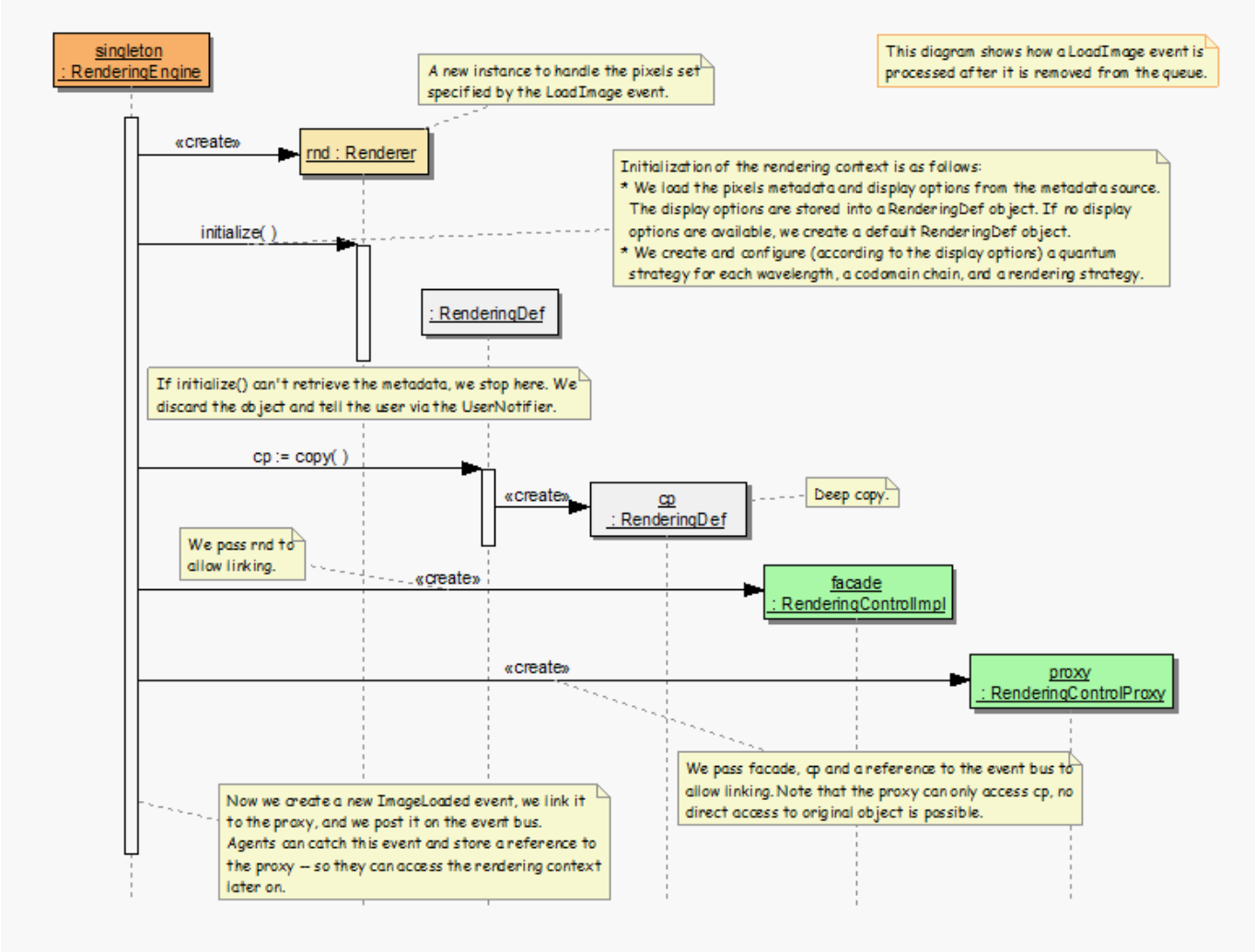
Calls which use Hibernate for the cross-database conversion can use the `org.hibernate.Session` interface.

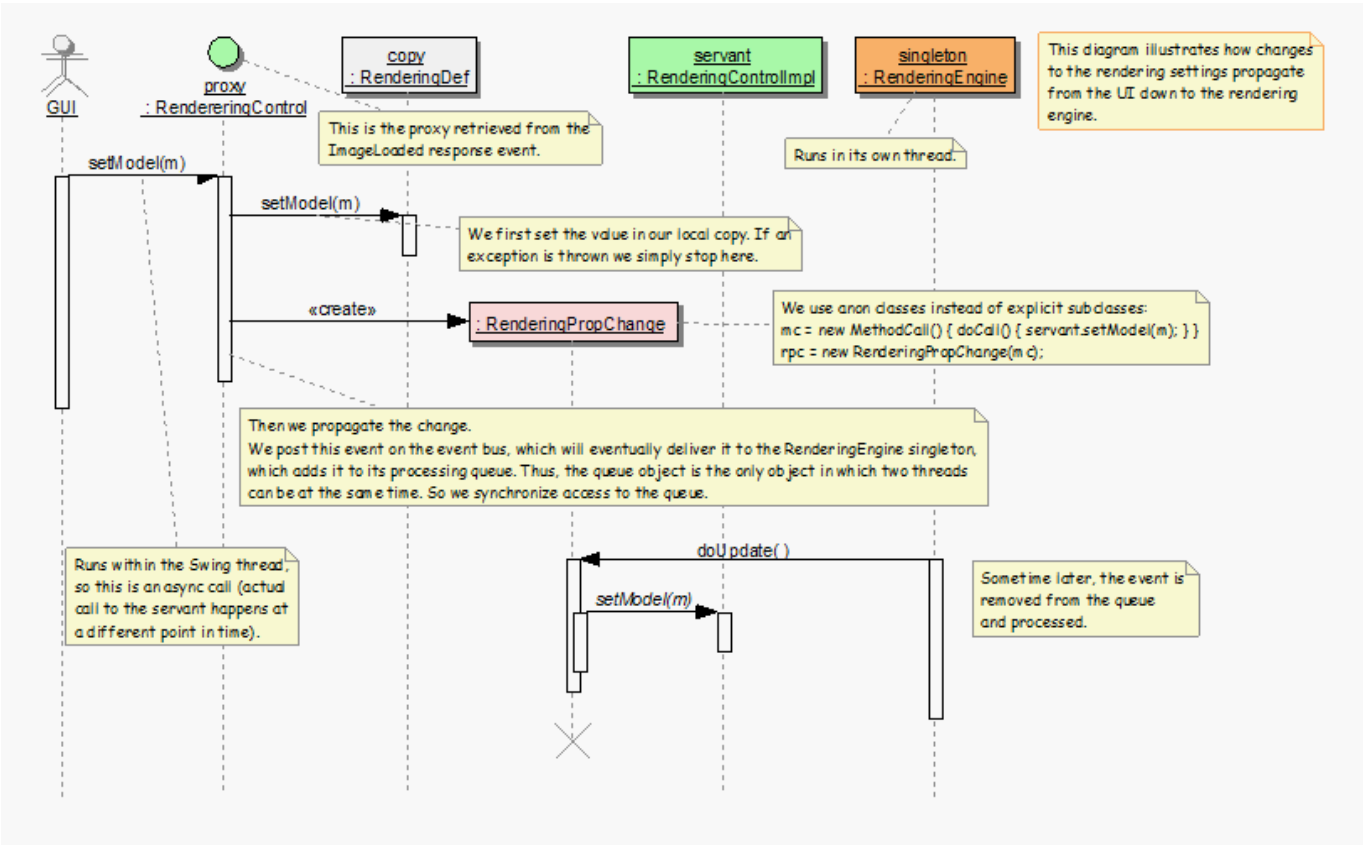
This documentation is for OMERO 4.4 and is no longer being updated, to see the documentation for the latest release, refer to http://openmicroscopy.org/site/support/omero/

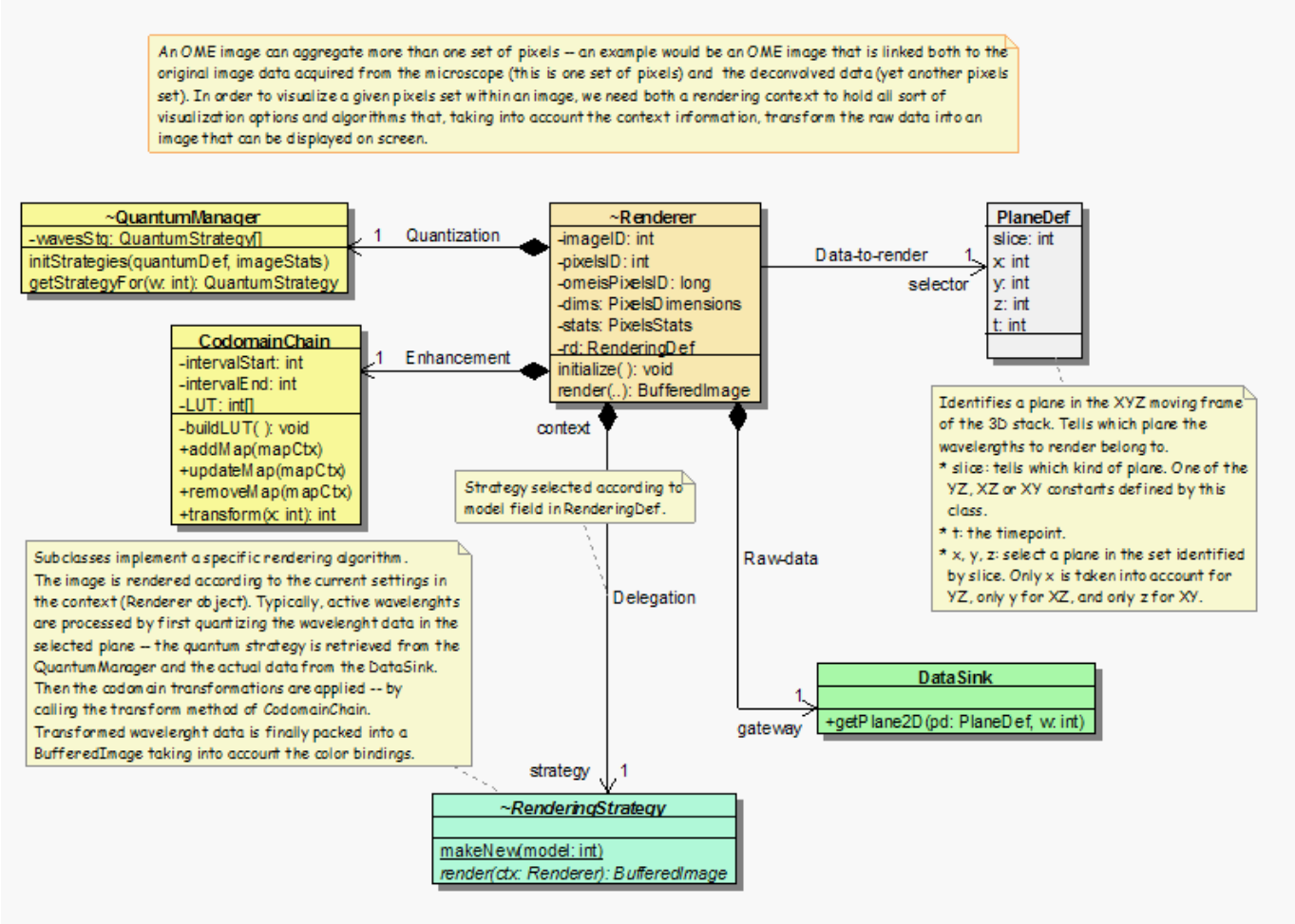
21.19 OMERO.fs

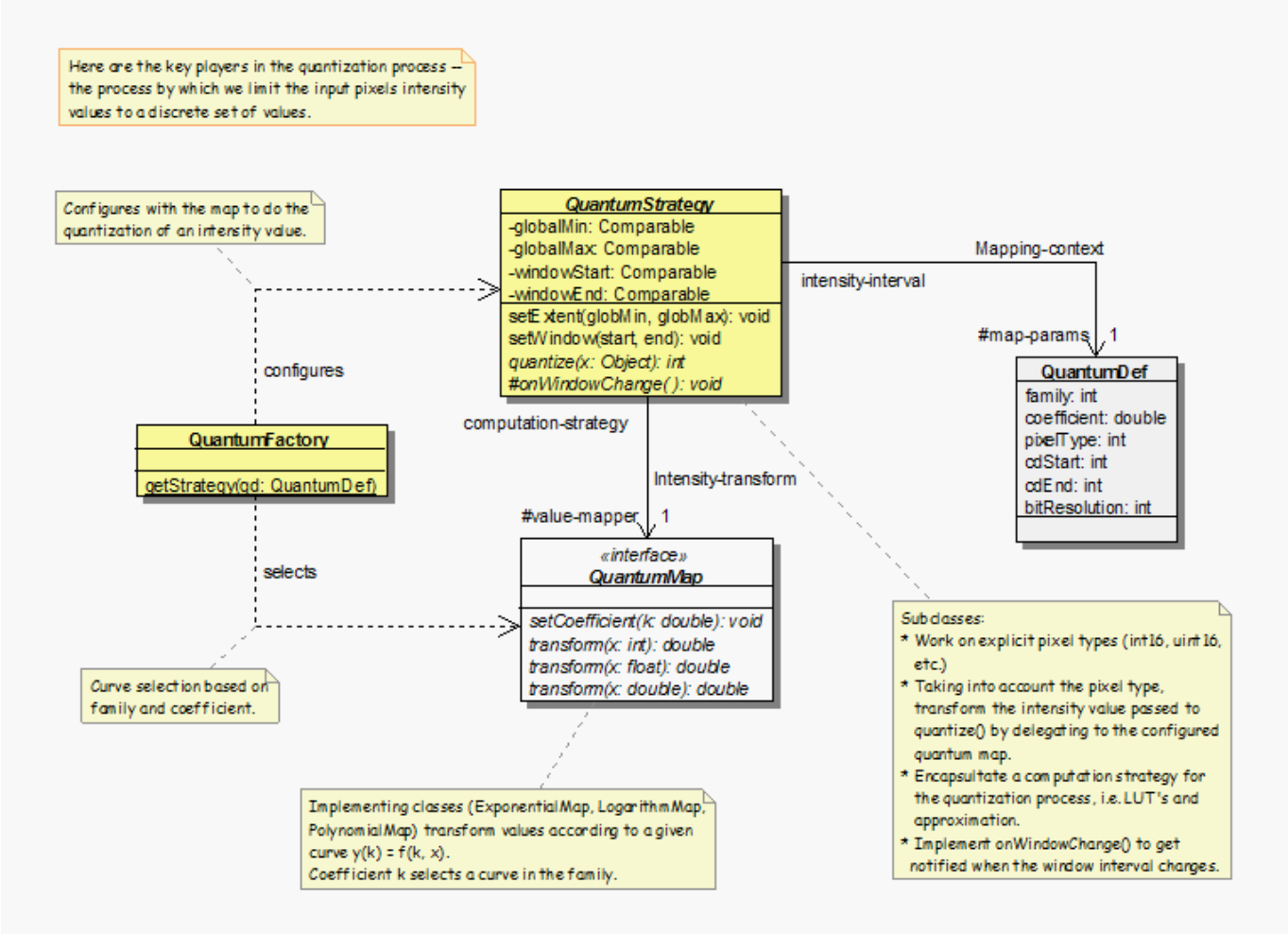
OMERO.fs is a series of on-going changes designed to improve the way an OMERO.server interacts with existing directories of acquired image data. These changes are currently being implemented almost exclusively in the OMERO 5 development line. In OMERO version 4.4, OMERO.fs consists of a single component:

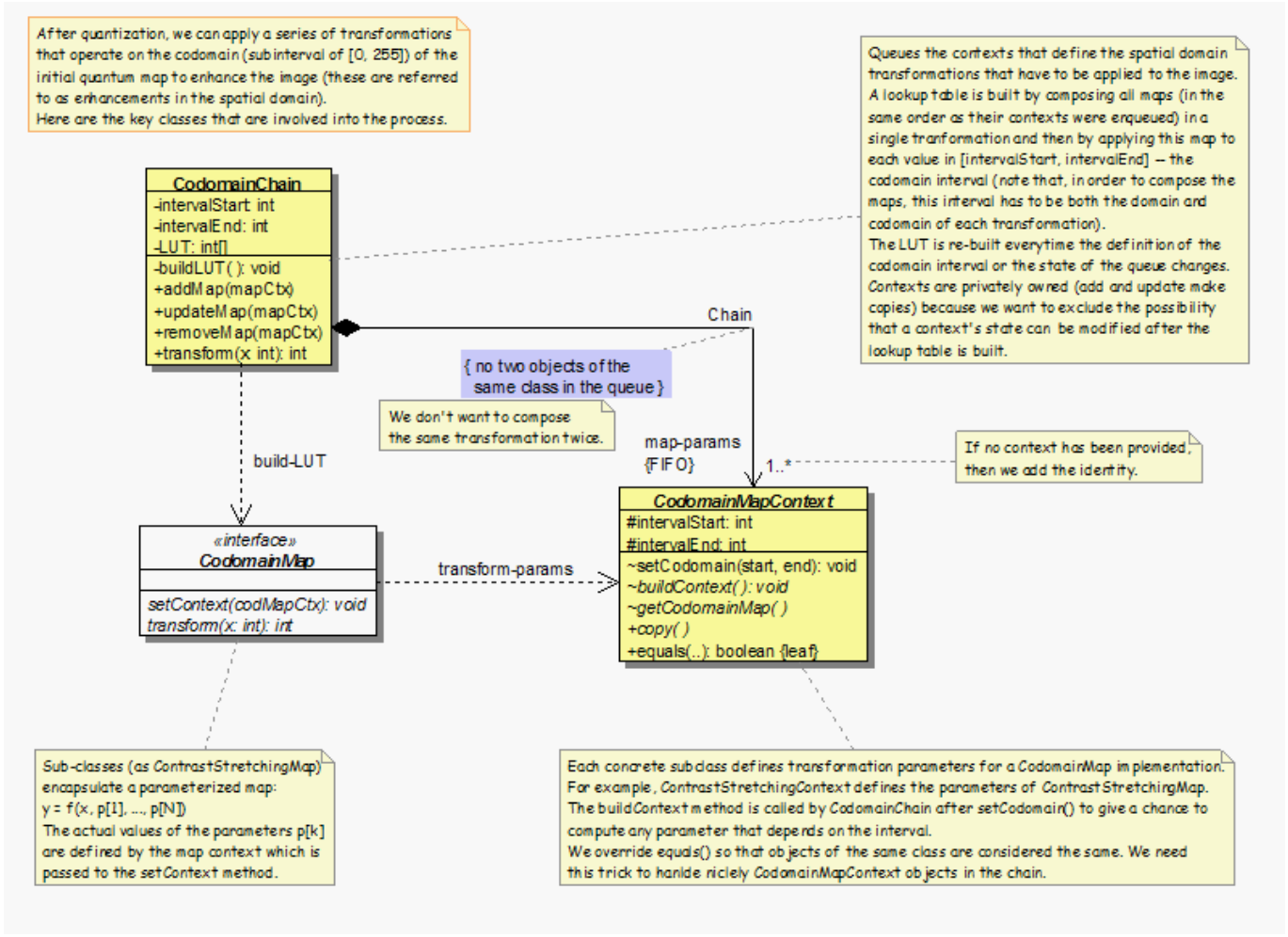
OMERO.dropbox is designed for watching a directory and kicking off an automatic import. The configuration of the DropBox system is covered on the *OMERO.dropbox* system administrator's page.

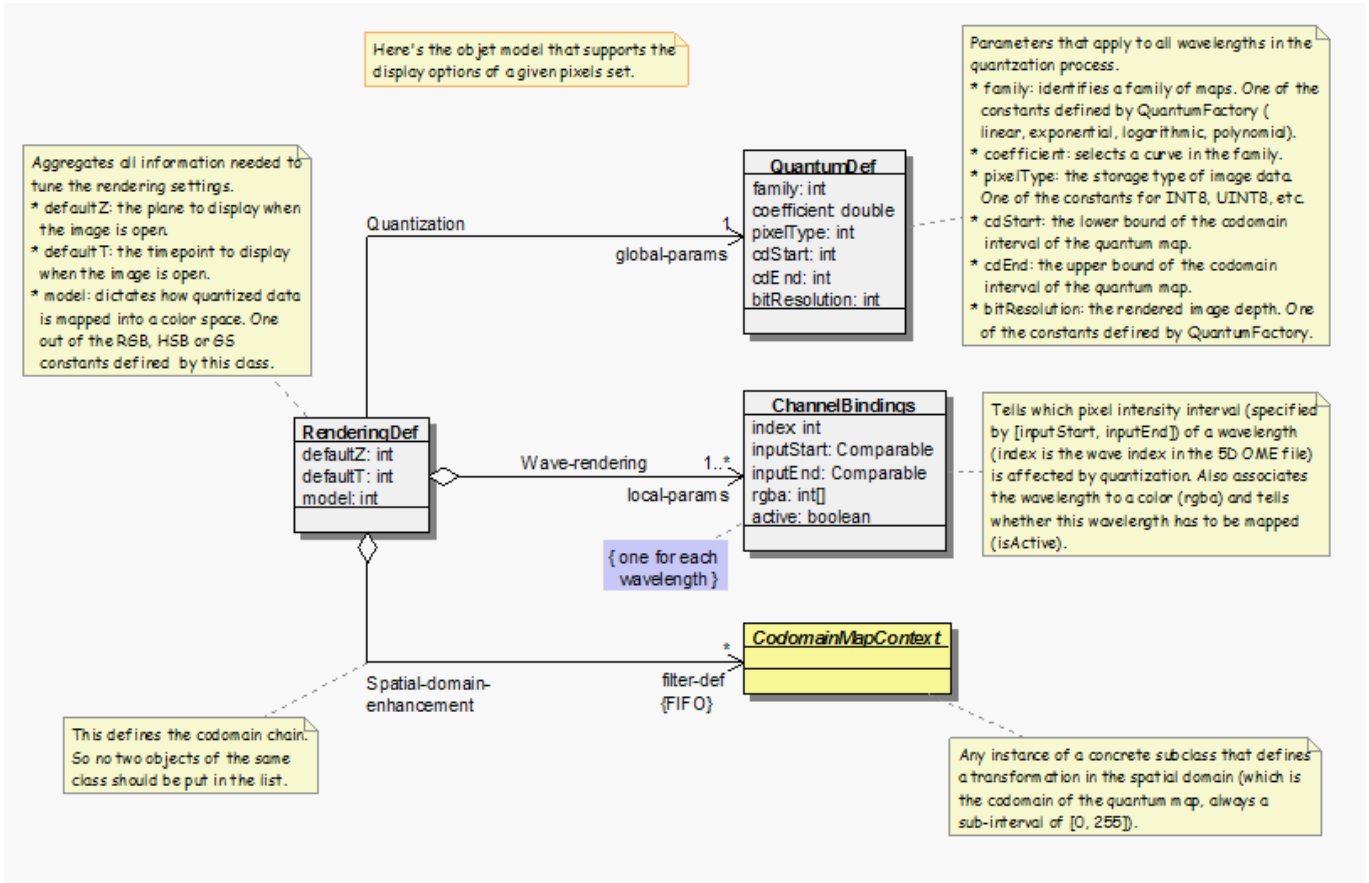

singleton
: RenderingEngine
A new instance to handle the pixels set specified by the LoadImage event.
This diagram shows how a LoadImage event is processed after it is removed from the queue.
«create»
rnd : Renderer
Initialization of the rendering context is as follows:
* We load the pixels metadata and display options from the metadata source. The display options are stored into a RenderingDef object. If no display options are available, we create a default RenderingDef object.
* We create and configure (according to the display options) a quantum strategy for each wavelength, a codomain chain, and a rendering strategy.
initialize( )
: RenderingDef
If initialize() can't retrieve the metadata, we stop here. We discard the object and tell the user via the UserNotifier.
cp := copy( )
«create»
cp
: RenderingDef
Deep copy.
We pass rnd to allow linking.
«create»
facade
: RenderingControlImpl
«create»
proxy
: RenderingControlProxy
We pass facade, cp and a reference to the event bus to allow linking. Note that the proxy can only access cp, no direct access to original object is possible.
Now we create a new ImageLoaded event, we link it to the proxy, and we post it on the event bus. Agents can catch this event and store a reference to the proxy -- so they can access the rendering context later on.

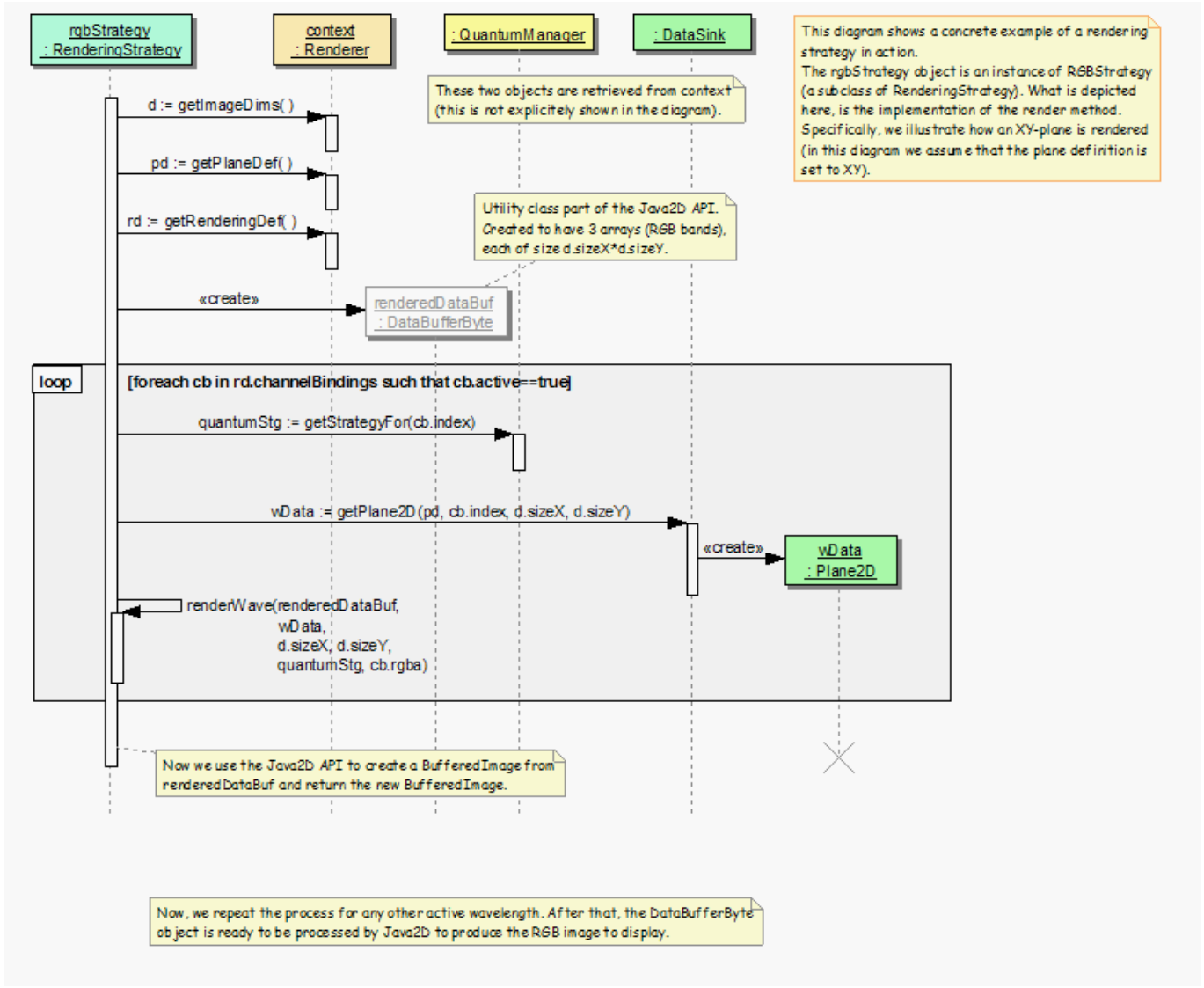

GUI
proxy
: RendereringControl
copy
: RenderingDef
servant
: RenderingControlImpl
singleton
: RenderingEngine
This diagram illustrates how changes to the rendering settings propagate from the UI down to the rendering engine.
This is the proxy retrieved from the ImageLoaded response event.
Runs in its own thread.
setModel(m)
setModel(m)
We first set the value in our local copy. If an exception is thrown we simply stop here.
«create»
: RenderingPropChange
We use anon classes instead of explicit subclasses:
mc = new MethodCall() { doCall() { servant.setModel(m); } }
rpc = new RenderingPropChange(mc);
Then we propagate the change.
We post this event on the event bus, which will eventually deliver it to the RenderingEngine singleton, which adds it to its processing queue. Thus, the queue object is the only object in which two threads can be at the same time. So we synchronize access to the queue.
Runs within the Swing thread, so this is an async call (actual call to the servant happens at a different point in time).
doUpdate( )
setModel(m)
Sometime later, the event is removed from the queue and processed.

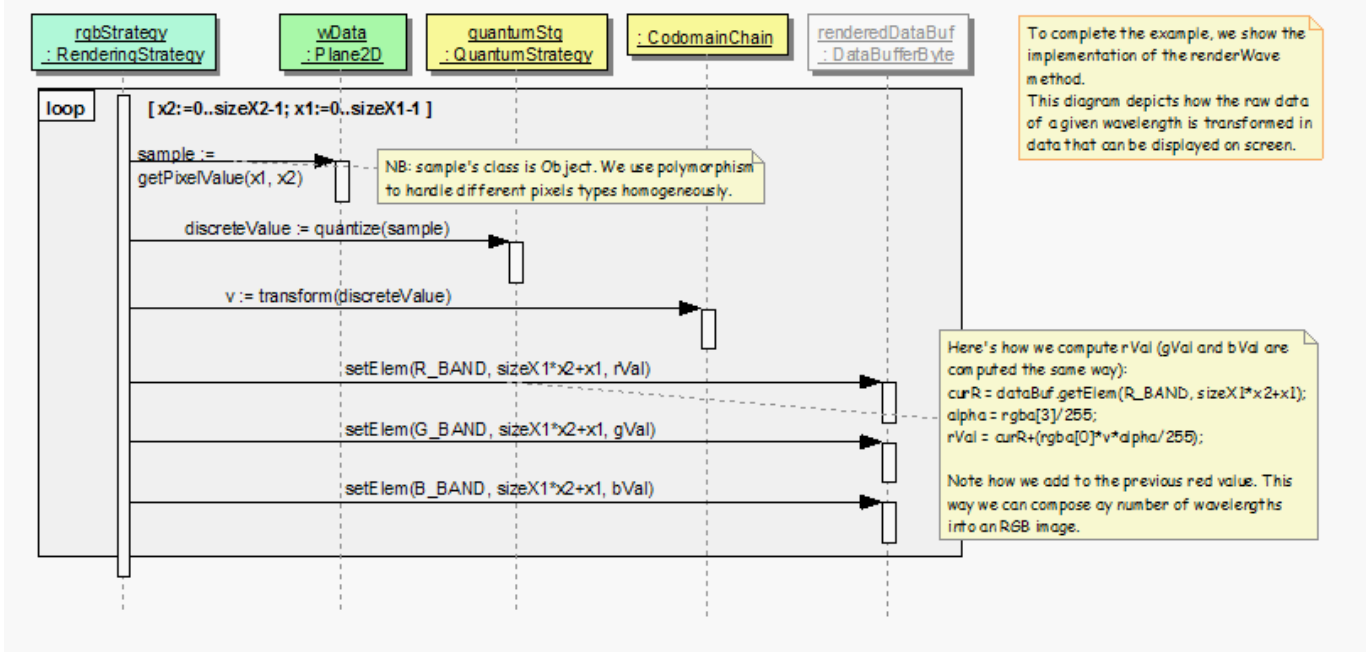

An OME image can aggregate more than one set of pixels -- an example would be an OME image that is linked both to the original image data acquired from the microscope (this is one set of pixels) and the deconvolved data (yet another pixels set). In order to visualize a given pixels set within an image, 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.
~QuantumManager
-wavesStg: QuantumStrategy[]
initStrategies(quantumDef, imageStats)
getStrategyFor(w: int): QuantumStrategy
1
Quantization
~Renderer
-imageID: int
-pixelsID: int
-omeisPixelsID: long
-dims: PixelsDimensions
-stats: PixelsStats
-rd: RenderingDef
initialize( ): void
render(..): BufferedImage
Data-to-render
1
selector
PlaneDef
slice: int
x: int
y: int
z: int
t: int
CodomainChain
-intervalStart: int
-intervalEnd: int
-LUT: int[]
-buildLUT( ): void
+addMap(mapCtx)
+updateMap(mapCtx)
+removeMap(mapCtx)
+transform(x: int): int
1
Enhancement
context
Identifies a plane in the XYZ moving frame of the 3D stack. Tells which plane the wavelengths to render belong to.
* slice: tells which kind of plane. One of the YZ, XZ or XY constants defined by this class.
* t: the timepoint.
* x, y, z: select a plane in the set identified by slice. Only x is taken into account for YZ, only y for XZ, and only z for XY.
Strategy selected according to model field in RenderingDef.
Subclasses implement a specific rendering algorithm.
The image is rendered according to the current settings in the context (Renderer object). Typically, active wavelenghts are processed by first quantizing the wavelenght data in the selected plane -- the quantum strategy is retrieved from the QuantumManager and the actual data from the DataSink.
Then the codomain transformations are applied -- by calling the transform method of CodomainChain.
Transformed wavelenght data is finally packed into a BufferedImage taking into account the color bindings.
Raw-data
Delegation
DataSink
+getPlane2D(pd: PlaneDef, w: int)
1
gateway
strategy
1
~RenderingStrategy
makeNew(model: int)
render(ctx: Renderer): BufferedImage


Here are the key players in the quantization process – the process by which we limit the input pixels intensity values to a discrete set of values.
Configures with the map to do the quantization of an intensity value.
QuantumStrategy
-globalMin: Comparable
-globalMax: Comparable
-windowStart: Comparable
-windowEnd: Comparable
setExtent(globMin, globMax): void
setWindow(start, end): void
quantize(x: Object): int
#onWindowChange( ): void
Mapping-context
intensity-interval
#map-params
1
QuantumDef
family: int
coefficient: double
pixelType: int
cdStart: int
cdEnd: int
bitResolution: int
configures
computation-strategy
QuantumFactory
getStrategy(qd: QuantumDef)
Intensity-transform
#value-mapper
1
selects
«interface»
QuantumMap
setCoefficient(k: double): void
transform(x: int): double
transform(x: float): double
transform(x: double): double
Curve selection based on family and coefficient.
Subclasses:
* Work on explicit pixel types (int16, uint16, etc.)
* Taking into account the pixel type, transform the intensity value passed to quantize() by delegating to the configured quantum map.
* Encapsulate a computation strategy for the quantization process, i.e. LUT's and approximation.
* Implement onWindowChange() to get notified when the window interval changes.
Implementing classes (ExponentialMap, LogarithmMap, PolynomialMap) transform values according to a given curve y(k) = f(k, x).
Coefficient k selects a curve in the family.


After quantization, we can apply a series of transformations that operate on the codomain (subinterval of [0, 255]) of the initial quantum map to enhance the image (these are referred to as enhancements in the spatial domain).
Here are the key classes that are involved into the process.
CodomainChain
-intervalStart: int
-intervalEnd: int
-LUT: int[]
-buildLUT( ): void
+addMap(mapCtx)
+updateMap(mapCtx)
+removeMap(mapCtx)
+transform(x: int): int
Queues the contexts that define the spatial domain transformations that have to be applied to the image. A lookup table is built by composing all maps (in the same order as their contexts were enqueued) in a single tranformation and then by applying this map to each value in [intervalStart, intervalEnd] – the codomain interval (note that, in order to compose the maps, this interval has to be both the domain and codomain of each transformation).
The LUT is re-built everytime the definition of the codomain interval or the state of the queue changes. Contexts are privately owned (add and update make copies) because we want to exclude the possibility that a context's state can be modified after the lookup table is built.
Chain
{ no two objects of the same class in the queue }
We don't want to compose the same transformation twice.
map-params
{FIFO}
1..*
If no context has been provided, then we add the identity.
build-LUT
«interface»
CodomainMap
setContext(codMapCtx): void
transform(x: int): int
transform-params
CodomainMapContext
#intervalStart: int
#intervalEnd: int
~setCodomain(start, end): void
~buildContext( ): void
~getCodomainMap( )
+copy( )
+equals(..): boolean {leaf}
Sub-classes (as ContrastStretchingMap) encapsulate a parameterized map:
y = f(x, p[1], ..., p[N])
The actual values of the parameters p[k] are defined by the map context which is passed to the setContext method.
Each concrete subclass defines transformation parameters for a CodomainMap implementation. For example, ContrastStretchingContext defines the parameters of ContrastStretchingMap. The buildContext method is called by CodomainChain after setCodomain() to give a chance to compute any parameter that depends on the interval.
We override equals() so that objects of the same class are considered the same. We need this trick to hanlde riclely CodomainMapContext objects in the chain.


Here's the objet model that supports the display options of a given pixels set.
Aggregates all information needed to tune the rendering settings.
* defaultZ: the plane to display when the image is open.
* defaultT: the timepoint to display when the image is open.
* model: dictates how quantized data is mapped into a color space. One out of the RGB, HSB or GS constants defined by this class.
Quantization
1
global-params
QuantumDef
family: int
coefficient: double
pixelType: int
cdStart: int
cdEnd: int
bitResolution: int
Parameters that apply to all wavelengths in the quantization process.
* family: identifies a family of maps. One of the constants defined by QuantumFactory (linear, exponential, logarithmic, polynomial).
* coefficient: selects a curve in the family.
* pixelType: the storage type of image data. One of the constants for INT8, UINT8, etc.
* cdStart: the lower bound of the codomain interval of the quantum map.
* cdEnd: the upper bound of the codomain interval of the quantum map.
* bitResolution: the rendered image depth. One of the constants defined by QuantumFactory.
RenderingDef
defaultZ: int
defaultT: int
model: int
Wave-rendering
1..*
local-params
ChannelBindings
index int
inputStart: Comparable
inputEnd: Comparable
rgba: int[]
active: boolean
Tells which pixel intensity interval (specified by [inputStart, inputEnd]) of a wavelength (index is the wave index in the 5D OME file) is affected by quantization. Also associates the wavelength to a color (rgba) and tells whether this wavelength has to be mapped (isActive).
{ one for each wavelength }
*
Spatial-domain-enhancement
filter-def
{FIFO}
CodomainMapContext
This defines the codomain chain. So no two objects of the same class should be put in the list.
Any instance of a concrete subclass that defines a transformation in the spatial domain (which is the codomain of the quantum map, always a sub-interval of [0, 255]).


rgbStrategy
: RenderingStrategy
context
: Renderer
: QuantumManager
: DataSink
This diagram shows a concrete example of a rendering strategy in action.
The rgbStrategy object is an instance of RGBStrategy (a subclass of RenderingStrategy). What is depicted here, is the implementation of the render method. Specifically, we illustrate how an XY-plane is rendered (in this diagram we assume that the plane definition is set to XY).
These two objects are retrieved from context (this is not explicitely shown in the diagram).
d := getImageDims( )
pd := getPlaneDef( )
rd := getRenderingDef( )
Utility class part of the Java2D API. Created to have 3 arrays (RGB bands), each of size d.sizeX*d.sizeY.
«create»
renderedDataBuf
: DataBufferByte
loop
[foreach cb in rd.channelBindings such that cb.active==true]
quantumStg := getStrategyFor(cb.index)
wData := getPlane2D(pd, cb.index, d.sizeX, d.sizeY)
«create»
wData
: Plane2D
renderWave(renderedDataBuf,
wData,
d.sizeX, d.sizeY,
quantumStg, cb.rgba)
Now we use the Java2D API to create a BufferedImage from renderedDataBuf and return the new BufferedImage.
Now, we repeat the process for any other active wavelength. After that, the DataBufferByte object is ready to be processed by Java2D to produce the RGB image to display.


rgbStrategy
: RenderingStrategy
wData
: Plane2D
quantumStg
: QuantumStrategy
: CodomainChain
renderedDataBuf
: DataBufferByte
To complete the example, we show the implementation of the renderWave method.
This diagram depicts how the raw data of a given wavelength is transformed in data that can be displayed on screen.
loop
[x2:=0..sizeX2-1; x1:=0..sizeX1-1]
sample :=
getPixelValue(x1, x2)
NB: sample's class is Object. We use polymorphism to handle different pixels types homogeneously.
discreteValue := quantize(sample)
v := transform(discreteValue)
setElem(R_BAND, sizeX1*x2+x1, rVal)
setElem(G_BAND, sizeX1*x2+x1, gVal)
setElem(B_BAND, sizeX1*x2+x1, bVal)
Here's how we compute rVal (gVal and bVal are computed the same way):
curR = dataBuf.getElem(R_BAND, sizeX1*x2+x1);
alpha = rgba[3]/255;
rVal = curR+(rgba[0]*v*alpha/255);
Note how we add to the previous red value. This way we can compose ay number of wavelengths into an RGB image.

Symbols

A

C

D

E

G

I

J

L

M

O

P

R

S

U

V