open microscopy environment

Java Server Documentation

Version alpha

(Work in progress)

Table of Contents

Chapter 1. Introduction

1.1. What?

The OME java server is a port of the perl code make use of existing object-relational mapping (ORM) tools.

1.2. Why?

Speed, ease of development, agility, alternative implementations

1.3. How?

Just what are these implementations? Well,...

1.1. Other info

```
LICENSE

    OME is distributed with the GNU Lesser General Public License
    (LGPL). You should be aware that any code submitted to the OME
    project will also fall under this license. If you have issues
    relating to licensing, want to sub license parts of the OME source
    code, or wish to discuss the licensing scheme please contact the
    OME core developers (ome-devel@lists.openmicroscopy.org.uk).

COPYRIGHT

    The OME source code, specification, documentation and database schema are,
    in their entirety Copyright (C) 2003 Open Microscopy Environment,
    Massachusetts Institute of Technology, National Institutes of Health,
    University of Dundee. As with the OME license, any questions related to the
    copyright, copyright holders or copyright scheme should be directed to the
    OME core developers (ome-devel@lists.openmicroscopy.org.uk).

DEVELOPER LOCATIONS

    The OME core developers are split between three main locations: OME Boston
    (MIT and Harvard; managed by Peter Sorger and Erik Brauner), OME Dundee
    (University of Dundee; managed by Jason Swedlow) and OME Baltimore (NIH;
    managed by Ilya Goldberg).

RESOURCES

    Most of the OME developer related documentation is either in the
    source tree (/doc and pod in the source files) or located on the
    CVS webserver (http://cvs.openmicroscopy.org.uk/).  There is also
    a documentation website available at
    http://docs.openmicroscopy.org.uk/.

LINKS

        The OME Project:  http://www.openmicroscopy.org/
        OME developers:   http://cvs.openmicroscopy.org.uk/
        Web based CVS:    http://cvs.openmicroscopy.org.uk/cvsweb/
        Documentation:    http://docs.openmicroscopy.org.uk
        Bug Tracker:      http://bugs.openmicroscopy.org.uk

MAILING LISTS

        OME developers:   ome-devel@lists.openmicroscopy.org.uk
```

Chapter 2. Getting Started

2.1. Building

As with any new framework, it takes a while to get into the code base. Let's start with building.

To work with the OMERO source base, you will need to have a fully working OME Database, a Java Development Kit, and a Servlet container, as well as various environment variables.

- Install Java and a Servlet Container (Tomcat, Jetty, Resin, or JBoss et al.) Warning: The Omero server side code requires Java 5 and Tomcat 5.5.x.
- Set JAVA_HOME to your JDK installation
- Set MAVEN_HOME to your Maven installation or alternatively to OMERO_HOME/lib/maven/ If you use the latter also run:

  OMERO_HOME/lib/maven/bin/install_repo.sh HOME_DIR/.maven/repository

  install_repo.bat is available for Windows users. Also, place MAVEN_HOME/bin/maven(.bat) on your PATH.
- Copy docs/examples/build.properties.example to OMERO_HOME or HOME; edit the properties for your site.

  Note: This file should _not_ be put under revision control!

Now you are ready to *build and install* OMERO. Run `maven bootstrap` to prepare the installation. Then run `maven install`to place all jars and the war file in your local maven repository. Copy the war file under

OMERO_HOME/components/server/target

to your servlet container. Enjoy!

Alternatively, follow the container-specific instructions below.

There are several methods to make working with a Tomcat instance simpler.

Edit the Tomcat section of your build.properties file (see HACKING.txt if you don't have one).

- Run "maven" from OMERO_HOME If maven is not on your path, alternatively run: "OMERO_HOME/lib/maven/bin/maven"
- cd to OMERO_HOME/components/wars/srv
- Run "maven tomcat:deploy"

2.2. Installing

If you are not building from source, but have downloaded the war (web-application resource) file then your work is a bit simpler.

To install the OMERO server, simply copy the war (web application resource) file to your servlet container. Once it is unpacked, edit:

/WEB-INF/classes/spring.properties
to connect to the database.

Once you start your servlet container, you can run the test suite against it.

For more specific instructions, please see the INSTALL_*.txt files in the docs directory.

Note: These instructions are for releases only. If you are working from subversion, please see BUILDING.txt to get things running.

2.3. Using

Building clients that access the Omero service is as easy as having omero-client.jar and omero-model.jar on your classpath.

Then create a ServiceFactory and use it to obtain a Service.

```
        ServiceFactory services = new ServiceFactory();
        HierarchyBrowsing proxy = services.getHierarchyBrowsingService();
```

Chapter 3. Server Design

There are several concepts which play a large role in the design of the server. These include: the DAO pattern, AOP cross-cutting, and dependency injection. The first of these is handled (currently) by Hibernate. The other two by Spring.

Chapter 4. Developing with/for the Java server

4.1. Server-side development

4.1.1. Quick How-To

- Add to common/src/ome/api interface "[API]" Run "maven jar:install" for common
- Add to server/src/ome/logic class "[API]Impl" Add to server/src/ome/dao interface "[API]Dao" Add to server/src/ome/dao/hibernate class "[API]DaoHibernate"
- Code. (This may include writing queries in a "[API]Queries.hbm.xml" file
- Update spring configuration in: server/web/WEB-INF/{services.xml,dao.xml}
- Write test by extending AbstractDepedencyInjectionSpringContextTests with proper getConfigLocations

4.2. Client-side development

If you are satisfied with working with the generated domain model objects directly, working with the server is nearly trivial. You simply need to supply connection paramters, either in a `spring.properties` on your classpath or through system variables. After that, all calls on any `ServiceFactory` object will return a functioning proxy.

If, however, the possibility instability of this worries you, it is straight forward to design an adapter around the existing model and API. To implement an adapter, you will need to define your own domain model objects and provide an implementation of the AdapterUtil iterfaces for your service. This will convert

Currently, there is no solution for adapting in the write direction. This is a result of the original intent of the server (read-only), and is currently on the TODO list.

4.3. Using Eclipse as an IDE

There are currently `.project` and `.classpath` files stored in subversion. Maven can reproduce them (for example, after changes to `project.xml`, but the existing files contain certain modifications that make working with the code base easier. You will need, however, certain classpath variables (`MAVEN_REPO`, `OMERO_HOME`, `USER_HOME`) to make them work.

More work needs to be done to make the Eclipse projects more useful. This will be completed at a later date.

Chapter 5. Testing

or *"Dependency injection, getConfigLocations, integration, oh my."*

5.1. Unit Testing

The unit testing framework is fairly simple. Only methods which contain logic written within the OME Java Server are tested. This means that framework functionality like remoting is *not* tested. Neither is DAO functionality; this is a part of integration testing. (see below)

Therefore, most of the code which is unit tested lies in the logic packages of the server component. This is done using jMock [http://jmock.org].

You can run the unit tests for any component from its directory by entering:

```
        maven test -Dmaven.test.mode=unit

        # or if you haven't changed the value of maven.test.mode simply:
        maven test
```

The same can be done for all components using:

```
        maven test-all
```

from the top-level directory.

5.2. Integration Testing

Integration testing is a bit more complex.

Because of there reliance on a database (which is not easily mockable), all DAO classes are tested in integration mode.

To run integration tests, use `-Dmaven.test.mode=integration` while calling the `test` and `test-all` maven targets mentioned under Unit Testing.

5.2.1. DbUnit Testing

Several special integration tests, based on the DbUnit [http://dbunit.sourceforge.net/] JUnit extension, are also included in the integration tests. This currently requires the creation of a special DB (specifically `"[your standard DB url]-test"`).

Currently, these tests will fail. Documentation on preparing these tests will be added later.

Chapter 6. Todo List

or, "Who what when?"

For a list of todo items, their priorities, assignees, and so forth please see TODO.html [./TODO.html].