UCSS Infrastructure : OMERO Installation

This page last changed on Mar 18, 2010 by arcierikw.

OMERO Installation for Mac OS X Snow Leopard(10.6)
Written by: Janek Claus and Kenneth Arcieri
National Institutes of Health (NIH) - National Institute of Child Health and Human Development (NICHD) - Unit on Computer Support Services (UCSS)

TODO:
Move "var" folder, which includes var/log, var/master, var/registry

Welcome to the step-by-step instructions for installing OMERO. This guide covers the installation for OMERO 4.1.1-Beta from December 2009. OMERO was initially not built for Mac OS X 10.6 and several incompatibilities started appearing when running OMERO. Most importantly Ice-3.3.1 is not originally built for 64-bit. Ice-3.4 is, but OMERO is not compatible with this version. So it was decided to build ICE 3.3 for 64 and start from the ground up.

Overall there are several major steps involved:

1. Install Postgre SQL
2. Install the Internet Communication Engine (ICE) and its dependencies
3. Install the Open Microscopy Environment (OMERO) and its dependencies

The following sections will go into detail for each step. Note that all libraries, applications and supporting files are configured and build from the source for the Mac OS X (10.6) environment.

Any questions or comments for this guide can be directed at:
kenneth.arcieri@nih.gov

Table of Contents:

Since this is a guide for Snow Leopard (Mac OS X, 10.6) it assumes that the following libraries are already installed (usually by default) and usable:

1. Python v2.6
2. Java v1.6
3. OpenSSL v0.9.81
4. bzip2 v1.0.5
5. Apache v2.2.13

In this guide we do everything from the command line. If you are not familiar with the vim text editor, please read up on its basic usage or use a text editor you are comfortable with.

Everything else will be explained below in the order we installed it. This setup was done using a fresh install of the operating system.

Required Libraries

The section lists all the libraries that will be required to build from the sources. We suggest you download them all at once, so you have them available for the installation instructions that follow.

If not otherwise mentioned download the sources by default.

- postgresql-8.4.2.tar.gz: http://www.postgresql.org/download/
- Ice-3.3.1.tar.gz: http://www.zeroc.com/download_3_3_1.html (Under "Source Distributions")
- ThirdParty-Sources-3.3.1.tar.gz: http://www.zeroc.com/download_3_3_1.html (Under "Source Distributions")
- omero-Beta4.1.1.tar.bz2 (binary districution): http://www.openmicroscopy.org/site/support/omero4/downloads
- hdf5-1.8.4.tar.gz: http://www.hdfgroup.org/downloads/
- tables-2.1.2.tar.gz: http://www.pytables.org/moin/Downloads
- libjpeg-8: http://www.ijg.org/files/jpegsrc.v8.tar.gz
- libpng-1.2.39: http://downloads.sourceforge.net/project/libpng/libpng-stable/1.2.39/libpng-1.2.39.tar.gz
- Freetype-2.3.12: http://sourceforge.net/projects/freetype/files/freetype2/2.3.12/freetype-2.3.12.tar.gz/download
- littleCMS-1.19: http://www.littlecms.com/lcms-1.19.tar.gz
- Python Imaging Library-1.1.7: http://effbot.org/downloads/Imaging-1.1.7.tar.gz
- matplotlib-0.99.1: http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-0.99.1/matplotlib-0.99.1.2.tar.gz/download
- mencoder: http://cvs.openmicroscopy.org.uk/snapshots/mencoder/mac/mencoder.zip
- flex-2.5.35: http://prdownloads.sourceforge.net/flex/flex-2.5.35.tar.gz?download
- Python Module for Apache-3.3.1: http://apache.ziply.com/httpd/modpython/mod_python-3.3.1.tgz
- Django-1.1.1: http://www.djangoproject.com/download/1.1.1/tarball/

File Placement Overview

The installation directories all follow the general Mac OS X installation guidelines. The top-level directories are subdivided for each minor version and patch. This allows future extensions and patches to be installed without disrupting the whole system or already built tools and libraries.

The outline of the directories is as follows:

```
+ TopLevelDirectory
  +- Current@ -> 3.6 (to the latest stable minor version)
  +- 3.5@ -> ./Versions/3.5.1 (to the latest stable patch version)
  +- 3.6@ -> ./Versions/3.6.8
  +- ...
  +- Versions
    +- 3.5.1/
    +- ...
    +- 3.6.8/
    +- ...
```

Here are the top-level directories for the libraries and tools:

Library or tool	Top-level directory
BerkeleyDB	/Library/Frameworks/BerkeleyDB
Expat	/Library/Frameworks/expat
flex	/Library/flex
Freetype	/Library/Frameworks/Freetype
HDF5	/Library/Frameworks/HDF5
ICE	/Library/Ice
libjpeg	/Library/Frameworks/libjpeg
libpng	/Library/Frameworks/libpng
littleCMS	/Library/Frameworks/lcms
MCPP	/Library/mcpp
mencoder	/Library/mencoder
OMERO	/Library/Omero
Postgre SQL	/Library/PostgreSQL

PostgreSQL Installation

PostgreSQL is an open-source object-relational database management system (ORDBMS) based on POSTGRES, Version 4.2, developed at the University of California at Berkeley Computer Science Department.

PostgreSQL Directory preparation

```
> sudo mkdir /Library/PostgreSQL
> sudo mkdir /Library/PostgreSQL/Versions
> sudo mkdir /Library/PostgreSQL/Versions/8.4.2
> cd /Library/PostgreSQL
> sudo ln -s Versions/8.4.2 8.4
> sudo ln -s 8.4 Current
```

PostgreSQL Building and installing

```
> cd {POSTGRE_SOURCE_DIR}
> ./configure --prefix=/Library/PostgreSQL/Versions/8.4.2
> make
> make check
> sudo make install
```

The tablespace check will fail during this process. It is a known bug at the time of this installation. Ignore it and proceed with the rest of the installation.

PostgreSQL User creation

Determine an unused user ID by running the following command and writing down the highest ID in the 500 category, then add 1 (one) to it to be used for the new user:

```
> dscl . -list /Users UniqueID | sort -n -k2
```

Create the user account in the local directory:

```
> sudo dscl . -create /Users/postgres
> sudo dscl . -create /Users/postgres UserShell /bin/bash
> sudo dscl . -create /Users/postgres RealName "Postgre SQL"
> sudo dscl . -create /Users/postgres UniqueID "502"  <-- Insert the previously found user ID!!
> sudo dscl . -create /Users/postgres PrimaryGroupID 20
> sudo dscl . -create /Users/postgres NFSHomeDirectory /Users/postgres
```

Set the password:

```
> sudo dscl . -passwd /Users/postgres {REPLACE WITH PASSWORD}
```

Create home directory:

```
> sudo createhomedir -c
```

PostgreSQL Data directory initialization

```
> sudo mkdir -p /var/lib/pgsql/data
> sudo chown postgres:staff /var/lib/pgsql/data
> su postgres -c '/Library/PostgreSQL/Current/bin/initdb -D /var/lib/pgsql/data'
```

You can test the installation:

```
> su - postgres
> /Library/PostgreSQL/Current/bin/postgres (don't start in the background)
in a new terminal:
> su - postgres
> /Library/PostgreSQL/Current/bin/psql -U postgres -l
```

All installed databases should now be listed. Shutdown the database using CTRL-C.

Post-Installation configuration

Some configuration options need to be set in the config file.

1. Create an original copy of the Postgre SQL configuration file

```
    > su - postgres
    > cd /var/lib/pgsql/data
    > cp postgresql.conf postgresql.conf.orig
    > vim postgresql.conf
     - uncomment 'logging_collector' and set it to 'on'
     - uncomment 'log_rotation_age' and set it to 0
     - uncomment 'log_rotation_size'
```

PostgreSQL Launch daemon configuration

The launch daemon is responsible for starting Postgre SQL during startup time.

1. The configuration needs to stored in a property list file in a specific location.

```
> cd /Library/LaunchDaemons
> sudo vim {YOUR_DOMAIN}.postgresql.plist
```

2. The following text needs to be entered into this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
       "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>Label</key>
   <string>{YOUR_DOMAIN}.postgresql</string>

    <key>UserName</key>
    <string>postgres</string>

    <key>Program</key>
    <string>/Library/PostgreSQL/Current/bin/postgres</string>

    <key>WorkingDirectory</key>
    <string>/Library/PostgreSQL/Current/bin</string>

    <key>EnvironmentVariables</key>
    <dict>
       <key>PGDATA</key>
       <string>/var/lib/pgsql/data</string>
       <key>PATH</key>
       <string>/Library/PostgreSQL/Current/bin:$PATH</string>
    </dict>

    <key>RunAtLoad</key>
    <true/>
</dict>
</plist>
```

3. Test that the process can be loaded successfully:

```
> sudo launchctl list | grep gov  --> should be empty
> sudo launchctl load {YOUR_DOMAIN}.postgresql.plist
> sudo launchctl list | grep gov
  --> should list the name '{YOUR_DOMAIN}.postgresql'
      with a valid process ID (the ID of the Postgre master process)
> sudo ps -ef | grep postgres
  --> check that the master process ID matches the
      ID recorded in the launch daemon list
```

4. Finally unload the daemon configuration and shutdown Postgre (it got loaded during loading of the plist because of the 'RunAtLoad' flag).

```
> sudo launchctl unload {YOUR_DOMAIN}.postgresql.plist
> kill {MASTER_PROCESS_ID}
```

5. Restart the server. **Congratulations, your Postgre SQL installation is now functional!**

Dependencies for ICE

Berkeley DB

An embeddable database

1. Patch the Berkeley DB as written in the README from ThirdParty-Sources-3.3.1.tar.gz
2. Configure the build from "{BERKLEY_SOURCE_DIR}/build_unix" directory:

```
> ../dist/configure --enable-cxx --enable-java --prefix=/Library/Frameworks/BerkeleyDB/Versions/4.6.21
> make
> sudo make install
```

3. Create version links:

```
> sudo ln -s /Library/Frameworks/BerkeleyDB/Versions/4.6.21 /Library/Frameworks/BerkeleyDB/4.6
> sudo ln -s /Library/Frameworks/BerkeleyDB/4.6 /Library/Frameworks/BerkeleyDB/Current
```

MCPP

A portable C/C++ preprocessor

1. Remove the lines 560-562 from "{MCPP_SOURCE_DIR}/src/internal.H":

```
560 #if HOST_HAVE_STPCPY
561 extern char *   stpcpy( char * dest, const char * src);
562 #endif
```

2. Configure the build from "{MCPP_SOURCE_DIR}/" directory:

```
> ./configure CFLAGS=-fno-common --enable-mcpplib --disable-shared --prefix=/Library/mcpp/
Versions/2.7.2
> make
> sudo make install
```

3. Create version links:

```
> sudo ln -s /Library/mcpp/Versions/2.7.2 /Library/mcpp/2.7
> sudo ln -s /Library/mcpp/2.7 /Library/mcpp/Current
```

Expat

A C library for parsing XML

1. Configure the build from "{EXPAT_SOURCE_DIR}/" directory:

```
> ./configure --prefix=/Library/Frameworks/expat/Versions/2.0.1
> make
> sudo make install
```

2. Create version links:

```
> sudo ln -s /Library/Frameworks/expat/Versions/2.0.1 /Library/Frameworks/expat/2.0
> sudo ln -s /Library/Frameworks/expat/2.0 /Library/Frameworks/expat/Current
```

ICE installation

ICE needs to be built in chunks. We only need the core libraries (cpp), the Java and Python bindings. The builds are self-contained for each of the binding and need be configured before the ICE distribution can be built.

Prepare directories

```
> cd /Library
> sudo mkdir -p Ice/Versions/3.3.1
> cd Ice
> sudo ln -s Versions/3.3.1/ 3.3
> sudo ln -s 3.3 Current
```

ICE-cpp preparation

```
> cd {ICE_SOURCE_DIR)/cpp
> modify config/Make.rules
  # set prefix to /Library/Ice/Versions/3.3.1
  # set embedded_runpath_prefix to /Library/Ice/3.3
  # uncomment OPTIMIZE
  # uncomment DB_HOME and set to /Library/Frameworks/BerkeleyDB/4.6
  # uncomment EXPAT_HOME and set to /Library/Frameworks/expat/2.0
  # uncomment MCPP_HOME and set to /Library/mcpp/2.7
```

ICE-Java preparation

```
> cd {ICE_SOURCE_DIR)/java/config
> cp build.properties build.properties.orig
> vim build.properties
  # set prefix to /Library/Ice/Versions/3.3.1 (same as for cpp)

  # In order for ICE for Java to compile the ICEGridAdmin it needs the JGoodies libraries.
  # This is a build time dependency only and does not affect the final build,
  # the dependencies specified for the build process will need to be specified again,
  # when running the ICEGridAdmin (which we don't intend). At the time the libraries
  # are needed for runtime they can be copied to a production location.

  # set jgoodies.forms to {3RD_PARTY_SOURCE_DIR}/forms-1.2.0/forms-1.2.0.jar
  # set jgoodies.looks to {3RD_PARTY_SOURCE_DIR}/looks-2.1.4/looks-2.1.4.jar
```

ICE-Python preparation

```
> cd {ICE_SOURCE_DIR)/py
> modify config/Make.rules
  # set prefix to /Library/Ice/Versions/3.3.1 (same as for cpp)
  # set embedded_runpath_prefix to /Library/Ice/3.3
  # uncomment OPTIMIZE
```

Configuration and install

Configure the Makefile to only build aforementioned modules: cpp, Java and Python:

```
> cd {ICE_SOURCE_DIR}
> cp Makefile Makefile.orig
> vim Makefile
  # in SUBDIRS remove cs,rb and php
  # same in CLEAN_SUBDIRS, DEPEND_SUBDIRS, INSTALL_SUBDIRS
```

Set environment and run make:

```
> export CLASSPATH=/Library/Frameworks/BerkeleyDB/4.6/lib/db.jar
> make              (builds all sources)
> sudo make install  (installs only the core module!)
> cd py
> sudo make install
> cd java
> sudo make install
```

Dependencies for OMERO.server

HDF5

Data model, library, and file format for storing and managing data.
Prepare directories:

```
> sudo mkdir -p /Library/Frameworks/HDF5/Versions/1.8.4
> cd /Library/Frameworks/HDF5
> sudo ln -s Versions/1.8.4 1.8
> sudo ln -s 1.8 Current
```

Build and install:

```
> cd {HDF5_SRCDIR}
> ./configure --prefix /Library/Frameworks/HDF5/Versions/1.8.4
> make
> make test
> sudo make install
```

Pytables

A hierarchical database package designed to efficiently manage very large amounts of data.
PyTables installs itself into the plugin system of the already installed Python, no directory creation needed.

```
> python setup.py build_ext --inplace --hdf5=/Library/Frameworks/HDF5/1.8
> python
>>> import tables
>>> tables.test()
>>> exit()
> sudo python setup.py install --hdf5=/Library/Frameworks/HDF5/1.8
```

OMERO.server Installation

OMERO.server User creation

Determine an unused user ID by running the following command and writing down the highest ID in the 500 category, then add 1 (one) to it to be used for the new user:

```
> dscl . -list /Users UniqueID | sort -n -k2
```

Create the user account in the local directory:

```
> sudo dscl . -create /Users/omero
> sudo dscl . -create /Users/omero UserShell /bin/bash
> sudo dscl . -create /Users/omero RealName "Omero"
> sudo dscl . -create /Users/omero UniqueID "503"  <-- Insert the previously found user ID!!
> sudo dscl . -create /Users/omero PrimaryGroupID 20
> sudo dscl . -create /Users/omero NFSHomeDirectory /Users/omero
```

Set the password:

```
> sudo dscl . -passwd /Users/omero {REPLACE WITH PASSWORD}
```

Create home directory:

```
> sudo createhomedir -c
```

OMERO Directory preparation

```
> sudo mkdir -p /Library/Omero/Versions/4.1.1
> cd /Library/Omero
> sudo ln -s Versions/4.1.1 4.1
> sudo ln -s 4.1 Current
```

The directory for the variable data needs to be created:

```
> sudo mkdir -p /var/lib/omero/data
> sudo mkdir -p /var/lib/omero/tmp
> sudo chown -R omero:staff /var/lib/omero
```

The directory for 'var' needs to be created and made accessible to OMERO. Unfortunately the location is hardcoded and can not be configured.

```
> sudo mkdir /Library/Omero/Versions/4.1.1/var
```

Copy the OMERO binary distribution into the destination location:

```
> sudo cp -R {OMERO_DIST_DIR) /Library/Omero/Versions/4.1.1
> sudo chown -R omero:staff /Library/Omero/Versions/4.1.1
```

OMERO.server Intialization

If you want to be able to administer the OMERO server manually as well (it will be setup as a daemon), I suggest setting the following:

```
export PATH=/Library/Ice/3.3/bin:$PATH
export PATH=/Library/PostgreSQL/8.4/bin:$PATH
export PYTHONPATH=/Library/Ice/3.3/python:$PYTHONPATH
export OMERO_TEMPDIR=/var/lib/omero/tmp
```

It is also a good idea to add these paths to the .profile of the omero user so you can su into that account and manage OMERO manually.

su into the omero account and then create the omero user for the database and initialize the Omero database:

```
> su - omero
> createuser -U postgres -P -D -R -S omero
> createdb -U postgres -O omero omero
> createlang -U postgres plpgsql omero
```

In the distribution directory generate the database initialization script, run it with psql and then logout back into the administrator account:

```
> bin/omero db script

> psql -U omero omero < OMERO4.1__0.sql
> logout
```

Omero Dropbox is looking for a specific directory. Unless the directory is there and created, OMERO FS will also fail:

```
> sudo mkdir /var/lib/omero/data/DropBox
> sudo chown omero:staff /var/lib/omero/data/DropBox
```

Due to a bug in Java for OS X, this next step might be needed to make the server run.
Check to see if the follow file exists:

```
> ls /Library/Preferences/com.apple.java.util.prefs.plist
```

If the listing returns nothing do the following.

Create a file with the following content and save it as "com.apple.java.util.prefs.plist":

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>/</key>
   <dict/>
</dict>
</plist>
```

Next the plist file and convert it to a binary plist:

```
> plutil -convert binary1 com.apple.java.util.prefs.plist
```

Finally move this file to its proper location:

```
> mv com.apple.java.util.prefs.plist /Library/Preferences/com.apple.java.util.prefs.plist
```

OMERO.server Configuration

There are two ways of going about configuring OMERO for use. One is to use the command line path to set configuration variables. The other is to create an xml file with the settings. In this guide we are going to use the command line method. The reason for this is because it creates a binary plist in "~/Library/Preferences/omero.prefs.default.plist" that will store all parameters set with the following command:

```
> bin/omero config set [parameter] [value]
```

First, log in as the omero user and tell OMERO about the data path:

```
> su - omero
> bin/omero config set omero.data.dir /var/lib/omero/data
```

Next, set the database information and logout:

```
> bin/omero config set omero.db.name omero
> bin/omero config set omero.db.user omero
> bin/omero config set omero.db.pass omero
> logout
```

Dependencies for OMERO.web

libjpeg

JPEG reference library
Read more: http://www.faqs.org/faqs/jpeg-faq/part1/section-1.html#ixzz0iYQRYvWj

Prepare directories:

```
> sudo mkdir -p /Library/Frameworks/libjpeg/Versions/8
> cd /Library/Frameworks/libjpeg
> sudo ln -s Versions/8 8
> sudo ln -s 8 Current
```

Build and install:

```
> cd {LIBJPEG_SRCDIR}
> ./configure --prefix=/Library/Frameworks/libjpeg/Versions/8
> make
> sudo make install
```

libpng

PNG reference library
Prepare directories:

```
> sudo mkdir -p /Library/Frameworks/libpng/Versions/1.2.39
> cd /Library/Frameworks/libpng
> sudo ln -s Versions/1.2.39 1.2
> sudo ln -s 1.2 Current
```

Build and install:

```
> cd {LIBPNG_SRCDIR}
> ./configure --prefix=/Library/Frameworks/libpng/Versions/1.2.39
> make check
> sudo make install
```

*NOTE: At the time of writing this libpng 1.4.1 was released, but matplotlib wouldn't compile with that release. Use the most stable 1.2.x version.

Freetype

Portable font engine.
Prepare directories:

```
> sudo mkdir -p /Library/Frameworks/Freetype/Versions/2.3.12
> cd /Library/Frameworks/Freetype
> sudo ln -s Versions/2.3.12 2.3
> sudo ln -s 2.3 Current
```

Build and install:

```
> cd {FREETYPE_SRCDIR}
> ./configure --prefix=/Library/Frameworks/Freetype/Versions/2.3.12
> make
> sudo make install
```

littleCMS

A CMM engine. It provides fast transforms between ICC profiles
Prepare directories:

```
> sudo mkdir -p /Library/Frameworks/lcms/Versions/1.19
> cd /Library/Frameworks/lcms
> sudo ln -s Versions/1.19 1.19
> sudo ln -s 1.19 Current
```

Build and install:

```
> cd {LCMS_SRCDIR}
> ./configure --prefix=/Library/Frameworks/lcms/Versions/1.19
> make
> make check
> sudo make install
```

Python Imaging Library

Image processing functionality support for many file formats
Prepare the setup.py file"

```
> cd {PIL_SRCDIR}
> vim setup.py
> edit:
> 37 JPEG_ROOT = libinclude("/Library/Frameworks/libjpeg/Versions/8")
> 40 FREETYPE_ROOT = libinclude("/Library/Frameworks/Freetype/Versions/2.3.12")
> 41 LCMS_ROOT = libinclude("/Library/Frameworks/lcms/Versions/1.19")
```

Build and install:

```
> cd {PIL_SRCDIR}
> python setup.py build_ext -i
> python selftest.py
> sudo python setup.py install
```

matplotlib

Plotting library for the Python programming language and its NumPy numerical mathematics extension.
Build and install:

```
> cd {MATPLOTLIB_SRCDIR}
```

Before the build we need to add some directories to the setupext.py file so it will compile correctly:

```
Around line 329 add the following:
  329     module.include_dirs.extend(incdirs)
  330     module.include_dirs.append('.')
+ 331     module.include_dirs.append('/Library/Frameworks/Freetype/Current/include')
+ 332     module.include_dirs.append('/Library/Frameworks/libpng/Current/include')
  333     module.library_dirs.extend(libdirs)
  334     module.library_dirs.append('.')
+ 335     module.library_dirs.append('/Library/Frameworks/Freetype/Current/lib')
+ 336     module.library_dirs.append('/Library/Frameworks/libpng/Current/lib')
```

Continue with the install:

```
> python setup.py build
> python setup.py install
```

mencoder

Command line video decoding, encoding and filtering tool
Prepare directories:

```
> sudo mkdir -p /Library/mencoder
```

Move the mencoder file from the zip file into this directory and then:

```
> sudo ln -s /Library/mencoder/mencoder /usr/local/bin/mencoder
```

flex

Fast lexical analyzer
Prepare directories:

```
> sudo mkdir -p /Library/flex/Versions/2.5.35
> cd /Library/flex
> sudo ln -s Versions/2.5.35 2.5
> sudo ln -s 2.5 Current
```

- NOTE : During this install the test-concatenated-options failed, I simply ignored it and the system still seems to work fine.

Build and install:

```
> cd {FLEX_SRCDIR}
> ./configure --prefix=/Library/flex/Versions/2.5.35
> make
> make check
> sudo make install
```

mod_python

Apache module that embeds the Python interpreter within the server
Modify {MOD_PYTHON_SRCDIR}/src/connobject.c
Change line 142:

```
!(b == APR_BRIGADE_SENTINEL(b) ||
```

To:

```
!(b == APR_BRIGADE_SENTINEL(bb) ||
```

Build and install:

```
> cd {MOD_PYTHON_SRCDIR}
> ./configure --with-apxs=/usr/sbin/apxs --with-flex=/Library/flex
> make
> sudo make install
```

After the install open your httpd.conf (Default on Snow Leopard is /private/etc/apache2/httpd.conf).
Find where to add modules and add the following line:

```
LoadModule python_module /usr/libexec/apache2/mod_python.so
```

OMERO.web Installation

First, check to see if the packaged web-server is set to "on-demand." This is found in the configuration file {OMERO_HOME}\etc\grid\default.xml (It is in the default.xml unless you have changed the name of your distribution.) Look for the following server-instance template and id. If the act is different then "on-demend," change it to "on-demand."

```
<server-instance template="ShellTemplate" id="Web" act="on-demand"/>
```

Create the directories that will store the web log files and the web database. In our installation we chose /var/lib/omero/weblog and /var/lib/omero/webdb/. In addition make these directories owned by omero:

```
> sudo mkdir -p /var/lib/omero/weblog
> sudo mkdir -p /var/lib/omero/webdb
> sudo chown omero:staff /var/lib/omero/weblog
> sudo chown omero:staff /var/lib/omero/webdb
```

As the omero user, run the omero web settings script so omero will create a custom_settings.py file to setup the django application.

```
> su - omero
> {OMERO_HOME}/Current/bin/omero web settings
> logout
```

Go to {OMERO_HOME}/Current/lib/python/ directory and remove the folder named "django". Next untar django-1.1.1 and move the "django" folder from that tar to the {OMERO_HOME}/Current/lib/python/ directory.

Copy the db.sqlite3 from {OMERO_HOME}/Current/lib/python/omeroweb/ to /var/lib/omero/webdb/

Change the ownership of the weblog and webdb directories to the apache web-user, in our case _www:

```
> sudo chown _www:_www /var/lib/omero/weblog
> sudo chown -R _www:_www /var/lib/omero/webdb
```

Now open up {OMERO_HOME}/Current/lib/python/omeroweb/settings.py:

```
Change around line 44:
DATABASE_NAME = 'db.sqlite3'  ->   DATABASE_NAME = '/var/lib/omero/webdb/db.sqlite3'

Change around line 160:
Comment out all LOGDIR with #
Add line:
LOGDIR = '/var/lib/omero/weblog'
```

Finally, tell the apache server to load the omero django application. This is done by inserting the following code into the httpd.conf file of your server:

```
<Location "/">
    SetHandler python-program
    PythonHandler django.core.handlers.modpython
    SetEnv DJANGO_SETTINGS_MODULE omeroweb.settings
    PythonDebug On

    PythonPath "['/Library/Ice/Current/python', '/Library/Omero/Current/lib/python', '/Library/Omero/Current/
lib/python/omeroweb'] + sys.path"
</Location>
```

OMERO Launch daemon configuration

Take the follow code and save it to /Library/LaunchDaemons/{YOUR_DOMAIN}.omero.plist
If everything is installed correctly and the paths in this file are correct, OMERO will start up when the computer starts

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
     <key>Label</key>
     <string>{YOUR_DOMAIN}.omero</string>
```

```
    <key>UserName</key>
    <string>omero</string>

    <key>ProgramArguments</key>
    <array>
        <string>{OMERO_HOME}/Current/bin/omero</string>
        <string>admin</string>
        <string>start</string>
    </array>

    <key>WorkingDirectory</key>
    <string>{OMERO_HOME}Current/bin</string>

    <key>EnvironmentVariables</key>
    <dict>
        <key>PATH</key>
        <string>/usr/bin:{ICE_HOME}/Current/bin:{POSTGRE_HOME}/Current/bin:{OMERO_HOME}/Current/
bin:$PATH</string>

        <key>PYTHONPATH</key>
        <string>{ICE_HOME}/Current/python::$PYTHONPATH</string>

        <key>OMERO_TEMPDIR</key>
        <string>/var/lib/omero/tmp</string>
    </dict>

    <key>RunAtLoad</key>
    <true/>
</dict>
</plist>
```

Congratulations, your finished a full OMERO installation! Enjoy!