EMAN 2 Reference Manual

INSERT NAMES

Contents

Chapter 1

Introduction

1.1 Abstract

EMAN 2 is an open-source Software for Single Particle Analysis and Electron Micrograph Analysis. EMAN2 is designed to handle both small, low-symmetry single particles and large, high symmetry viruses. It uses more object-oriented technologies for better modular design, has convenient and flexible interfaces to promote extendibility, emphases easy and efficient use in Python, and many new capabilities.

1.2 Goals

1. EMAN2 should have a generic way to read/write various electronic microscopy image formats.

    - Support all electron-microscopy image formats supported in EMAN
    - Read an arbitrary convex 2D/3D region of any image format
    - Separate reading/writing of image headers from image data
    - Support averaging and shrinking when reading images

2. EMAN2 should make it easy to use and develop image filters.

    - It should be simple and easy to define a new filter in either C++ or Python.
    - Allow users to define new filters that accept arbitrary numbers of parameters in an arbitrary order
    - Utilize existing filters defined in EMAN1 system.
    - Filters should be able to be pipelined.
    - Each filter is identified with a meaningful name

3. The same design goals for filters should also apply to EM image aligners, image comparators, image averagers, image 3D reconstructors.

4. EMAN2 should have a generic way to handle image translation, rotation, Euler angles. It should support basic geometry operations like vector and matrix.

5. EMAN2 should allow different definitions of CTFs.

6. EMAN2 should have a modular design to support different FFTW library at compile time.

7. EMAN2 should support data processing on an arbitrary convex region of an image.

8. EMAN2 should support a logging mechanism similar to that in EMAN1.

9. EMAN2 should be designed for easy regression tests.

10. EMAN2 should be fully documented using Doxygen style.

11. EMAN2 should support elegant integration with Phoenix software:

    - Support data interchange with Phoenix CCTBX.
    - Implement basic image reconstruction tasks used in Phenix GUI environment.

Chapter 2

Installation

2.1 Required Libraries / Programs

The following libraries are required for EMAN2 installation (the libraries should be installed as shared-object libraries where applicable):

2.1.1 FFTW

Version 2.1.3+
Available at: http://www.fftw.org/

To install fftw from source use either configure option:

% ./configure –enable-static=no –enable-shared=yes –enable-float –enable-type-prefix

OR

% ./configure –enable-shared=yes –enable-float

Followed by:

% make

2.1.2 GSL - GNU Scientific Library

Version: 1.3+
Available at: http://www.gnu.org/software/gsl/

Installation is very straight forward:

% ./configure
% make

2.1.3 Boost

Version: 1.32 or lower
Available at: http://www.boost.org

(NOTE: EMAN2 does not currently support Boost versions above 1.32.)

1. Installing Boost requires Boost.Jam. Executables and source code for jam can be found at the Boost website.

Installing Boost requires the user to identify a particular toolset to use during compliation. Most UNIX systems will probably use the ”gcc” toolset; visit http://www.boost.org/more/getting_started.html#Tools for a complete listing.

% bjam ”-sTOOLS=gcc” install

Header files from the Boost installation (located in the ”boost” subdirectory of the Boost installation (ex. /boost_1_32_0/boost)) must now either be added to the compilers path or copied into an existing location on the path in a subdirectory /boost.

One possibility for this might be: % cp -r boost /usr/include/boost

2.1.4 CMake

Version: 2.0.6+
Available at: http://www.cmake.org

Executables for several platforms are available; source code can also be used for custom installations.

2.2 Optional Libraries / Programs

- To read/write HDF5 image, use hdf5 (http://hdf.ncsa.uiuc.edu/HDF5). (NOTE: HDF5 1.6.4 has some API compatibility issue and it doesn’t work with EMAN2 yet.)

- To read TIFF image, use libtiff (http://www.libtiff.org)

- To read PNG image, use PNG (http://www.libtiff.org)

For development the following libraries/programs are required (see 3.13 for installation help):

- Python (version 2.2+) (http://www.python.org)

- Boost Python (version 1.32-) (http://www.boost.org)

- Numerical Python Numpy (version 22.0+) (http://www.pfdubois.com/numpy)

2.3 Quick Installation

Suppose you have source code eman2.tar.gz

1. % cd $HOME

   % mkdir -p EMAN2/src

   % cd EMAN2/src

   % gunzip eman2.tar.gz

   % tar xf eman2.tar

2. % mkdir build

   % cd build

3. % cmake ../eman2

   % make

   % make install

4. set up login shell
   for csh/tcsh, put the following to your .cshrc or .tcshrc file:

   setenv EMAN2DIR $HOME/EMAN2
   setenv PATH $EMAN2DIR/bin:${PATH}
   setenv LD_LIBRARY_PATH $EMAN2DIR/lib
   setenv PYTHONPATH .:$HOME/EMAN2/lib

   for bash in .bashrc add:

   export EMAN2DIR=$HOME/EMAN2
   export PATH=$PATH:$EMAN2DIR/bin
   export LD_LIBRARY_PATH=$EMAN2DIR/lib
   export PYTHONPATH=$PYTHONPATH:$HOME/EMAN2/lib

2.4 Advanced Installation

If your libraries (fftw, gsl, hdf, etc) are not found by Quick Installation, or if you want to change the compilation options, the following steps help:

1. follow the first 2 steps in Quick Installation.

2. If your libraries are not installed at the default places, setup the related environment variables:

   - fftw ⟶ FFTWDIR
   - gsl ⟶ GSLDIR
   - tiff ⟶ TIFFDIR
   - png ⟶ PNGDIR
   - hdf5 ⟶ HDF5DIR
   - python ⟶ PYTHON_ROOT and PYTHON_VERSION

3. % ccmake ../eman2

   - type 'c' if it asks about "CMAKE_BACKWARDS_COMPATIBILITY".

- make necessary changes for compilation flags.
  - developers will probably want to set BOOST-LIBRARY to a Boost.Python object file (ex. libboost_python-gcc-1_32.so)
- Then type 'c', and type 'g'.

4. % make

   % make install

2.4.1 Platform Dependent Optimization

In CMake Configuration, enable the following option for your platform:

- Athlon: ENABLE_ATHLON
- Opteron: ENABLE_OPTERON
- Mac G5: ENABLE_G5

2.4.2 Generating the Latest Documentation

1. Install doxygen (version 1.4.3+, http://www.doxygen.org)
2. Run "sh ./makedoc.sh" from the "EMAN2/src/eman2/" directory to generate the documentation in the "EMAN2/src/eman2/doc/".

2.5 Notes for Developers

1. For Emacs users, please add the following line to your $HOME/.emacs: (setq default-tab-width 4)
2. Ensure Boost.Python is installed
3. EMAN2 uses Pyste (http://www.boost.org/libs/python/pyste/) to wrap C++ into python. Here is the way to install Pyste:

   (a) get boost python.
   % cd libs/python/pyste/install
   % python setup.py install

   (b) install elementtree

   (c) install GCCXML

   (d) for boost 1.32.0, apply a patch for PYSTE. (Contact EMAN2 developers for the patch.)

4. To generate new boost python wrapper, run

% cd eman2/libpyEm

% ./create_boost_python

5. Windows Installer EMAN uses ”Nullsoft Scriptable Install System” (http://nsis.sourceforge.net/) to generate the windows installer. It also uses ”HM NIS Edit” (http://hmne.sourceforge.net/) as the editor.

Chapter 3

Developer's Guide

3.1 Complete Reference Guide

See the Complete Reference Guide (/html/index.html) for a full listing of all classes and functions included in EMAN2.

3.2 Development Policy FAQ

Steve Ludtke, 11/26/2004

Introduction

This document summarizes the basic policy for EMAN2 development, especially on how to contribute your code to EMAN2. The intened audiences are EMAN2 developers.

Q. What is the current eman2 development model? Anybody with cvs access can change anything, or is there a "gatekeeper" who adds patches to the current code, and thus has ultimate authority over what gets into the program? What I mean is who decides whether a given change is desirable or not? Do you care if something that you are philosophically opposed to gets added to the code?

A. There aren't THAT many people with CVS access right now, so it isn't a major issue yet. As it stands now, anyone with CVS access is permitted to check changes in. Certainly anyone is allowed to add completely new routines, like new processors, new reconstruction routines, etc.

Q. Is there a policy about changing code submitted by somebody else? Similarly, is there anything that prevents (or facilitates) having several groups working on the same piece of code.

A. CVS automatically allows people to work on the same code simultaneously. If two people make conflicting changes (ie - change the same line of code), then the person who checks their changes back into CVS last has to go through the 'merging' process before CVS allows their changes to be committed. ie -

1. cvs checkout by A and B
2. A and B make conflicting changes to a particular file
3. A checks his changes back in
4. B tries to check his changes in, gets a message saying the code has already been changed, and he must resolve conflicts first
5. B does a cvs update, which will put BOTH versions of the changed code into the appropriate file. B must then manually update the code to reflect a single changed version.
6. B checks his changes in.

Note that, a record still exists of A's changes, so if B changes something in a bad way, it is still possible to check out A's version. Regarding changing someone elses code, I am certainly concerned about the issue of someone 'fixing' something and unintentionally breaking it. For now, please check with me before you spend time changing an existing piece of code. If there is a good reason to leave it as it is, you can simply make a different routine with the desired functionality. However, if something is really broken, or can be substantially improved, I'm all in favor of this happening. Just drop me a quick message explaining what you intend. For example, the background fitting routine you described in this message was never actually completed, and did not work, so putting in a functional version is a welcome addition.

Q. What's the policy of changing someone else's code?

A. I agree in general that changing someone elses code should only be done with permission of the original author. I would (for now) still like to act as a clearinghouse for this sort of activity, because there may be issues that the original author isn't even aware of. It is always possible to find out who wrote a particular routine through CVS, but this can be a bit annoying to accomplish. Right now we put author stamps at the top of each source code file.

Q. Is it clear who wrote a particular piece of code?

A. It is always possible to find out who wrote a particular routine through CVS, but this can be a bit annoying to accomplish. Right now we put author stamps at the top of each source code file. Starting now, why don't we set a policy that each new function/method that is added to EMAN2 should have an author/date stamp in the docstring for the method. I don't think this needs to extend to small changes within a routine. It is always possible to check CVS for detailed changes.

Q. User docs currently seem to be quite thin. Is that by design? Moreover, do the auto-generated docs adequately allow developers to determine how and by what a particular chunk of code is used. Also, is there a simple way to find out whether a given procedure used by multiple pieces of code and if yes, by which ones?

A. Well, there aren't really any user docs yet because there aren't really any user programs yet. There are just a few that we've started writing over the last few weeks. These should all be internally documented, ie e2pdb2mrc –help will provide some documentation, but this needs to be improved, and we do need to develop a script to collect this information into web pages. Right now our effort is mostly focused on programming, so the docs have also focused that way.

Q. What is the reason for having monolithic source files (such as processor.cpp) instead of having separate source files for each distinct piece of code? There are advantages to both solutions...

A. Simply to prevent having too many source files scattered around. If every Processor was in an individual file, there would be a tremendous number of files. As any particular file gets too large, we generally split it into a few smaller pieces. Since CVS takes care of the changes, and the code for a particular function can be localized within the source file, there aren't any major problems with this approach. One problem with the individual file method is that it tends to dramatically increase build and install times. This is a somewhat arbitrary policy.

Q. What is the policy on "stolen" code (such as that taken from Numerical Recipes)?

A. Clearly taking obvious copyrighted code is a BAD thing, especially when it has been made clear that such use is not permitted. However, I think NRC has actually relaxed their polices a bit recently. In most cases equivalent publicly available code can be found. We have already adopted GSL (Gnu Scientific Library), Opt++ and FFTW as dependencies. So routines from any of those toolkits can be used freely. If there are other toolkits you find you need, you need to talk to me before adding new dependencies to EMAN2 as a whole. More dependencies are generally a bad thing.

Q. May one incorporate pure C (or even pure Fortran) routines, particularly numerical routines, into eman2?

A. No Fortran. This is strictly due to the added compilation difficulties it poses. Pure C is a bit trickier, I certainly have no objections to C-like C++, but to work in the framework, the 'functions' need to at least be defined as static methods in some classes, for example, in Util class or EMUtil class.

Q. Is there a specification somewhere that distinguishes fatal from recoverable errors?

A. We are trying to adopt try/except exception processing for errors. The details for this are still being worked out. Fatal errors should be avoided if possible in the libraries, but when they occur, we should have a central routine to log the error and exit (EMAN1 did). This is in flux right now I'd say.

Q. Are operations on arrays / matrices defined anywhere in the code? Similarly, why does the EMData class define an image as a low-level C chunk of memory, with the user constantly performing on-the-fly index calculations?

A. Using C++ abstractions for the low level image storage (like an array of arrays) generally imposes a substantial speed penalty. There IS a higher level abstraction for accessing the image data get_value_at, set_value_at, etc. These are generally inlined functions, so the ones that don't check limits are pretty much the same speed as doing the index calculations yourself. Currently there aren't any routines for treating EMData objects as vectors/matrices/tensors, but this could be done. Liwei did make a matrix class, but this is used strictly for transformation matrices (3x3 or 4x4). GSL has a large set of matrix math routines. The main reason we haven't settled on anything is that we haven't completely decided on a particular linear algebra suite. We could use Numeric Python array on the python side, but this wouldn't provide C++ functionality. We could use GSL, but then we need to expose all of the appropriate routines on the python side, or we could decide to go full bore and adopt something like Lapack. This is still open for debate...

Q. Is there a testing policy or framework?

A. We would like to have a full set of unit-tests, so every method in EMAN2 will have a corresponding test. We are working on this. Bugfixes are still a higher priority right now. If you guys are actively working on EMAN2, I would strongly suggest doing frequent CVS updates. Wen and I have begun writing user-level programs now, and finding many bugs, which are rapidly getting fixed. Still, there are probably quite a few problems.

Q. How does eman2 report results (such as a goodness-of-fit, etcetera) to the user when manipulating images? The LOG facility appears to exist mainly for error/warning reporting. Some commands can produce copious output, foe example the ramp removal can print the coeffs of the fit. Should they be printed, and if yes, where?

A. Here is what I would propose:

1. In general, low level algorithms should not produce textual output aimed at the user. They should return appropriate values, then the user-level program can decide what to display.

2. If you really need, at least optionally, to produce output from within a C++ algorithm, use the LOG mechanism. By changing the LOG priority before making the function call, the higher level program can then decide whether the output should be generated or not. Lots of printing in high performance algorithms can have a severe impact on performance, so this should be avoided where possible. 3.6

3. most user level programs should provide a –verbose=<int> option to set the verbosity level. The user level program would then do appropriate things to implement this. This would generall be on a 0-9 scale with 0 being completely or almost completely silent and 9 being extremely verbose. In many cases only one or two levels may exist.

4. Note that image objects have an attribute mechanism, which can be used to store relevant, but indirect results. For example, say you called a routine which aligns one 2D image to another. The resulting aligned image might have attributes set indicating the alignment parameters and the goodness of fit.

5. A note to 4, above, we are gradually adopting the PDB/EBI standard attribute dictionary for many parameters, so we may require that algoritm specific attributes would be named in such a way that they are unique.

formatted by Liwei Peng on 11/30/2004

3.3 Coding Style

1. Introduction

   - This document summarizes the basic coding and naming style in EMAN2. The intended audiences are EMAN2 developers.
   - This document is not a programming tutorial. You may refer the references at the end of document for good books.

2. Overview

   (a) Coding:

      i. EMAN2 follows the GNU coding style with minor changes. We use indent at http://www.gnu.org/software/indent/indent.html for beautifying EMAN2 code.

   (b) Naming:

      i. All source code files use lower cases.
      ii. All classes and data types use uppercase in the first letter;
      iii. All functions use lower cases with '_' if necessary.

3. indent HowTo

   (a) install indent. (for linux, rpm is available from standard distribution.)
   (b) save file .indent.pro to your home directory.
   (c) say you have a file called "foo.C", run indent like this: indent foo.C
   (d) because indent is designed for C code, it is not perfect for C++ code. Read your new source and fix the following possible errors:

      i. change 'const const' to 'const'

4. Comments

   Use **Doxygen** JavaDoc style:

```
/** Brief description which ends at this dot. Details follow here.\\
 */\\
class Test\\
{\\
public:\\
/** The constructor's brief description in one line.\\
 * A more elaborate description of the constructor.\\
 */\\
Test();\\
\\
/** do some test.\\
 * @author Liwei Peng <lpeng@bcm.tmc.edu>\\
 * @date 1/20/2005\\
 * @param low the low threshold.\\
 * @param high the high threshold.\\
 * @return 0 if do_test succeeds; 1 if do_test fails.\\
 */\\
int do_test(float low, float high);\\
\\
/** Calculate the sum of an array.\\
 * @param[in] data Data array\\
 * @param[in] nitems Number of items in the array.\\
 * @param[out] sum The sum of the array.\\
 */\\
void calc_sum(int[] data, int nitems, int *sum);\\
\\
}\\
```

5. Samples
   (a) emdata.h
   (b) emdata.cpp
6. References

- Doxygen: `http://www.stack.nl/~dimitri/doxygen/`

Last modifiedby Liwei Peng (lpeng@bcm.tmc.edu)

3.4 Using Factory Classes

EMAN has many functions that implement specific algorithms to do image processing. These algorithms are relatively independent and a user chooses which one to use. EMAN2 the following factory classes are defined to manage these algorithms:

- **Processor** (image processor)
- **Aligner** (2D/3D image alignment)
- **Averager** (averaging a set of images)
- **Cmp** (comparing 2 same-size Images)
- **Projector** (3D image projection)
- **Reconstructor** (3D image reconstruction)

3.5 Exceptions

3.5.1 Using Exceptions

- Here is an example on throwing an exception:

```
vector <float>EMData::calc_fourier_shell_correlation(EMData * with){
 if (!with) {
  throw NullPointerException("NULL input image");
 }

 if (!EMUtil::is_same_size(this, with)) {
  LOGERR("images not same size");
  throw ImageFormatException( "images not same size");
 }
//...
}
```

- Here is an example on catching all possible exception

```
void foo()
{
    EMData* e1 = new EMData();
    EMData* e2 = new EMData();
    try {
        e1->read_image("test1.mrc");
        e2->read_image("test2.mrc");
        vector<float> v = e1->calc_fourier_shell_correlation(e2);
    }
    catch (E2Exception & exception) {
        printf("%s\n", exception.what());
```

```
    }
}
```

- Here is an example on catching a specific exception:

```
void foo()
{
    EMData* e1 = new EMData();
    EMData* e2 = new EMData();
    try {
        e1->read_image("test1.mrc");
        e2->read_image("test2.mrc");
        vector<float> v = e1->calc_fourier_shell_correlation(e2);
    }
    catch (_NullPointerException & exception) {
        printf("%s\n", exception.what());
    }
}
```

  Note the "_" before _NullPointerException.

3.5.2 Existing Exception Types

- See the complete EMAN2 Reference Guide

3.5.3 Defining New Exception Classes

- A XYZ Exception class is defined in the following way:
  - It will extend E2Exception class.
  - The class is named _XYZException.
  - The class has a function to return its name "XYZException".
  - A macro called "XYZException" is defined to simplify the usage of _XYZException class.
- Here is the code for NullPointerException.

```
class _NullPointerException : public E2Exception {
  public:
        _NullPointerException(const string& file = "unknown",
                              int line = 0,
                              const string& desc_str = "")
            : E2Exception(file, line, desc_str) {}
```

```
        const char *name() const{ return "NullPointerException"; }
    };
#define NullPointerException(desc) _NullPointerException(__FILE__,\
__LINE__,desc)
```

3.6 Using the Log Class

EMAN2 includes a built in class to handle writing log messages. It is highly recommended that all textual output be handled by this class.

3.6.1 Typical Usage

It is sometimes necessary to specify what output level of the message should be. There are 4 log levels to choose from

- *ERROR_LOG* - used for error messages
- *WARNING_LOG* - used for warning messages
- *DEBUG_LOG* - used for debugging messages
- *VARIABLE_LOG* - highly detailed debugging messages

The default value used is *ERROR_LOG*; to change the level use:

```
Log::logger()->set_level(Log::NEW_LOG_LEVEL);
```

Next, output the log message using:

```
Log::logger()->OUTPUT_FUNC;
```

where OUTPUT_FUNC is one of the following:

- ***error****(string)* - log an error message
- ***warn****(string)* - log a warning message
- ***debug****(string)* - log a debugging message
- ***variable****(string)* - log a very detailed debugging message

3.7 Adding Components

There are two general procedures for adding new components to EMAN2. The first option involves editing templates in order to integrate the new component into the core while the second option involves directly altering the existing core files.

It is generally recommended that components should be added first using templates and later directly added to existing core if desired. When testing or refining a new components build times are much faster if templates are used.

- General Overview for Using Templates:

  1. Edit the **template.h** and **template.cpp** files for the type of component that is being added
  2. From EMAN2/src/build run:
     % make

3.7.1 Processors

Using Templates:

Located in the plugin directory of the EMAN2 source (i.e. EMAN2/src/eman2/plugins) are various template files including **processor_template.h** and **processor_template.cpp**. These are the files that will be used for new processor installation. Begin by editing **processor_template.h**.

1. Change the occurrences of ”XYZ” in ”XYZProcessor” with the name of the new processor
   - Don’t forget to change the string in get_name() to the name of the processor (this is the name that will be used to call the processor)
2. Edit the string in get_desc() with a brief description of the processor. Place a more detailed descriptions elsewhere such as before the class or before the functions (see 3.3 for coding style information)
3. Define the processor’s parameters in get_param_types()
   - A description string can be added as a third param to the TypeDict::put() function to describe the variables
4. In the class constructor of FilterFactorExt uncomment the line ”Factory <Processor> ::add(&dProcessor::NEW);”

Now edit **processor_template.cpp**

- In processor() add the implementation code of the new processor.

- The existing template has sample code showing how to access the variables that where defined in get_para_types()
- Note that the sample code included in the template is enclosed in a conditional statement that essentially causes all of the code tobe skipped.

Finally rebuild EMAN:

```
% cd ../../build
% make
```

The new processor should now be available using the name that was specified in get_name().

Adding Directly to the Core:

If the new processor code has already been created using the supplied templates, then adding to the core can be done as follows:

1. Open processor.h in src/eman2/libEM
2. Copy the class you defined in processor_template.h and paste it in the file
3. Open processor.cpp (also located in src/eman2/libEM)
4. Copy and paste the class from processor_template.cpp to processor.cpp. Header file include statements may also need to be copied.
5. In the template class Factory located in the begining of processor.cpp add a line ”force_add(&newProcessor::NEW)” where ”newProcessor” is the name of the processor class that is being added
6. Rebuild EMAN2

The instructions for adding a new processor without first using templates go as follows:

1. In src/eman2/libEM open processor.h
2. Towards the end of the file there is an example class called XYZProcessor. Follow the first 3 steps listed in ”Using Templates”.
3. Open processor.cpp and write an implementation for the process() function that was just defined in processor.h
4. Repeat the last two steps (5 and 6) from the template installation

3.7.2 Aligners

The instructions are the same as those for adding a new processor (3.7.1) except the files of interest are aligner_template.h and aligner_template.cpp.

Also, when adding to the core, aligner.h and aligner.cpp should be altered.

3.7.3 Averagers

The instructions are the same as those for adding a new processor (3.7.1) except the files of interest are averager_template.h and averager_template.cpp.

Also, when adding to the core, averager.h and averager.cpp should be altered.

3.7.4 Image Comparator

The instructions are the same as those for adding a new processor (3.7.1) except the files of interest are cmp_template.h and cmp_template.cpp.

Also, when adding to the core, cmp.h and cmp.cpp should be altered.

3.7.5 IO Functionality

The instructions are the same as those for adding a new processor (3.7.1) except the files of interest are io_template.h and io_template.cpp.

Also, when adding to the core, *****NEED TO CHECK THIS***** should be altered.

3.7.6 Reconstructors

The instructions are the same as those for adding a new processor (3.7.1) except the files of interest are reconstructor_template.h and reconstructor_template.cpp.

Also, when adding to the core, reconstructor.h and reconstructor.cpp should be altered.

3.8 Adding Documentation

The majority of the documentation for EMAN2 is made using LaTeX (see the LaTeX homepage for more information). No preamble should be included in the new documentation; preamble information can removed either manually or using the python script rm_preamble.py (see 3.8.2 for more information).

Documentation can be created using a WYSIWYG LaTeX editor such as LyX or TeXmacs. When using these utilities do not create title pages, abstract sections, or tables of content as this will cause integration problems with the rest of documentation.

The recommended steps for adding new documentation go as follows:

1. Create a new stand alone document (see 3.8.1 for standards information)

2. Remove the preamble information from the new file (3.8.2)
3. Edit TOC.tex (3.8.3)
4. Run "make" to create the new documentation file

3.8.1 Document Standards

Please do not include the following sections/commands in new documentation:

- \title,\maketitle
- \tableofcontents

Edit the existing title and table of content information instead.

3.8.2 Removing Preambles

The preamble of a LaTeX document is used to specify various global parameters of the document. In a stand alone document, all commands coming before and including \begin{document} are part of the preamble. There is also a \end{document} at the very end of the document that should be removed.

Use the "%" character at the start of a line to comment it out.

The python script rm_preamble.py can usually be used to automatically remove the preamble and other tags that should not be used.

% python rm_preamble.py *file1 file2* ...

The output will be placed in the *file1*.tex, *file2*.tex, ...

3.8.3 Editing TOC.tex

TOC.tex contains the table of contents information for this document. Add "\input{*new file name*}" in the appropriate chapter to insert a new documentation file. Of course a new chapter can always be added using the "\chapter{*new chapter name*}" command.

3.8.4 Adding LaTeX Packages

The package inclusion tag for the documentation is located in EMAN2.tex. Add the package that is needed to the \usepackage command.

3.9 Image Header Attribute Naming Conventions

EMAN2 uses a unique name to reference each image header attribute. The following table lists the conventions:

Attribute Name	Data Type	Attribute Description
apix_x	float	pixel x size in
apix_y	float	pixel y size in
apix_z	float	pixel z size in
nx	int	image x-direction size in number of pixels
ny	int	image y-direction size in number of pixels
nz	int	image z-direction size in number of pixels
origin_row	float	origin location along row (x) in number of pixels
origin_col	float	origin location along column (y) in number of pixels
origin_sec	float	origin location along section (z) in number of pixels
minimum	float	minimum pixel density value in the image
maximum	float	maximum pixel density value in the image
mean	float	mean pixel density value in the image
sigma	float	sigma pixel density value in the image
square_sum	float	sum of the squares of each pixel density value in the image
mean_nonzero	float	mean pixel density value in the image, ignoring pixel whose value =0.
sigma_nonzero	float	sigma pixel density value in the image, ignoring pixel whose value =0.
kurtosis	float	kurtosis pixel density value in the image
skewness	float	skewness pixel density value in the image
orentation_convention	string	Euler orientation Convention, ”EMAN”, ”MRC”, etc
rot_alt	float	EMAN convention: alt
rot_az	float	EMAN convention: az
rot_phi	float	EMAN convention: phi
datatype	int	pixel data type in the image physical file (e.g., float, int, double, etc). This is an enumeration type.
is_complex	int	1 if this image is a complex image. 0 if it is a real image.
is_ri	int	1 if this is a complex image and is in real/imaginary format; 0 otherwise.
avgnimg	int	If the image is the result of averaging, avgnimg is the number of images used in averaging. Otherwise, avgnimg is 0.
label	string	user-defined label for this image. It is like ’name’ in EMAN1.
image_filename	string	the physical image filename where this image comes from.
image_index	int	image number in file pointed to by path.
DM3.exposure_number	int	DM3 image only. exposure number.
DM3.exposure_time	double	DM3 image only. exposure time in seconds.

Continued on next page

Attribute Name	Data Type	Attribute Description
DM3.zoom	double	zoom
DM3.antiblooming	int	antiblooming
DM3.magnification	double	Indicated magnification
DM3.frame_type	string	frame_type
DM3.camera_x	int	DM3 image only. camera size along x direction in number of pixels.
DM3.camera_y	int	DM3 image only. camera size along y direction in number of pixels.
DM3.binning_x	int	DM3 image only. Binning #0
DM3.binning_y	int	DM3 image only. Binning #1
MRC.nlabels	int	MRC image only. Number of comment labels in a MRC image.
MRC.labelN	string	MRC image only. The Nth label, where N is an integer from 1 to 10.
LST.filenum	int	LIST image only, file number in FileSystem server
LST.reffile	string	LIST image only, reference file
LST.refn	int	LIST image only, reference image number
LST.comment	string	LIST image only, comment
micrograph_id	string	Micrograph ID.
particle_center_x	float	x coordinate of particle location in the micrograph
partcle_center_y	float	y coordinate of particle location in the micrograph
ctf_dimension	int	CTF dimension, nD, where n is 1, 2, or 3.

Table 3.1: Image Attribute Namimg

3.10 Testing Framework

EMAN2 uses python unit test and regression test as its testing framework. For features not testable or not easily to be tested in python, they will be tested in C++.

3.10.1 Testing Guidelines

- Each unit test must be self contained.
- Each test module (.py file) should be named as ”test_” + featurename.
- Each test method should be named as ”test_” + method-name.
- unittest module should be used to do the unit testing. doctest is discouraged.

3.10.2 Test Example

- Example of a unit test and a regression test:

```
import unittest
from test import test_support

class MyTestCase1(unittest.TestCase):

    # Only use setUp() and tearDown() if necessary

    def setUp(self):
        ... code to execute in preparation for tests ...

    def tearDown(self):
        ... code to execute to clean up after tests ...

    def test_feature_one(self):
        # Test feature one.
        ... testing code ...

    def test_feature_two(self):
        # Test feature two.
        ... testing code ...

    ... more test methods ...

class MyTestCase2(unittest.TestCase):
    ... same structure as MyTestCase1 ...

... more test classes ...

def test\_main():
    test\_support.run\_unittest(MyTestCase1,
                              MyTestCase2,
                              ... list other tests ...
                             )

if __name__ == '__main__':
    test_main()
```

- Here is a more detailed example:

```
import random
import unittest

class TestSequenceFunctions(unittest.TestCase):
```

```
    def setUp(self):
        self.seq = range(10)

    def tearDown(self):
# do clean up here

    def testshuffle(self):
        # make sure the shuffled sequence does not lose any elements
        random.shuffle(self.seq)
        self.seq.sort()
        self.assertEqual(self.seq, range(10))

    def testchoice(self):
        element = random.choice(self.seq)
        self.assert_(element in self.seq)

    def testsample(self):
        self.assertRaises(ValueError, random.sample, self.seq, 20)
        for element in random.sample(self.seq, 5):
            self.assert_(element in self.seq)

if __name__ == '__main__':
    unittest.main()
```

- Asserting Values and Conditions

  The crux of each test is a call to assertEqual() to check for an expected result; assert_() to verify a condition; or assertRaises() to verify that an expected exception gets raised.

3.11 CVS Help

CVS is a version control system. Using it, you can record the history of sources files, and documents. For detailed information, you may refer http://www.gnu.org/software/cvs/.

This document gives the typical usage of CVS under Unix/Linux platforms. The intended audience are EMAN2 developers.

- Before you use CVS, set up environmental variable CVSROOT. For EMAN/EMAN2, set up the following in your shell startup script:
  - for csh/tcsh

    % setenv CVS_RSH ssh

    % setenv CVSROOT "blake.3dem.bioch.bcm.tmc.edu:/usr/local/CVS/CVS"

- for bash/sh/zsh

  % export CVS_RSH=ssh

  % export CVSROOT="blake.3dem.bioch.bcm.tmc.edu:/usr/local/CVS/CVS"

NOTE: The following supposes you put EMAN2 under $HOME/EMAN2.

- To check out EMAN2 source code, run

  % cd $HOME/EMAN2/src

  % cvs co eman2

- To add new files, run

  % cd $HOME/EMAN2/src/eman2

  % cd YOUR-DIRECTORY

  % cvs add YOUR-FILES

  % cvs commit

- To remove files, run

  % cd $HOME/EMAN2/src/eman2

  % cd YOUR-DIRECTORY

  % cvs remove YOUR-FILES

  % cvs commit

- To check in modified existing files, run

  % cd $HOME/EMAN2/src/eman2

  % cvs ci

- To update your source code to the latest version in the CVS tree, run

  % cd $HOME/EMAN2/src/eman2

  % cvs update

  NOTE: read the output from the above command. If you see merging conflicts, you must resolve them first.

3.12 Miscellaneous

3.12.1 Reading/Writing Images in Python

Example: Converting from IMAGIC to MRC or SPIDER:

```
from EMAN2 import *
e = EMData()
e.read_image(\u201cproj.img\u201d)
e.write_image(\u201cproj.mrc\u201d)
    # output image format is determined by file extension
e.write_image(\u201cproj.spi\u201d, 1, SPIDER)
    # explicitly specifiy the output image format
```

3.12.2 Processors Usage

Example: Using Processors in Python

```
from EMAN2 import

e = EMData()
e.read_image("test1.mrc")
e.process("eman1.math.sqrt")
e.process("eman1.threshold.binaryrange", {"low" : 5.2, "high" : 10})
e.write_image("output.mrc")
```

3.12.3 Reconstructors Usage

Example: Using Reconstructors in Python

```
from EMAN2 import *
import math

e1 = EMData()
e1.read_image(TestUtil.get_debug_image("samesize1.mrc"))

e2 = EMData()
e2.read_image(TestUtil.get_debug_image("samesize2.mrc"))

r = Reconstructors.get("back_projection")
r.set_params({"size":100, "weight":1})
r.setup()
r.insert_slice(e1, Transform(EULER_EMAN, 0,0,0))
r.insert_slice(e2, Transform(EULER_EMAN, math.pi/2,0,0))

result = r.finish()
result.write_image("reconstructor.mrc")
```

3.12.4 Using Pyste

EMAN2 uses Pyste to automatically parse C++ code to generate boost python wrappers. To use Pyste:

1. Install Pyste libraries/tools:
   (a) Pyste in boost library
   (b) elementtree
   (c) gccxml
2. Create or modify the pyste file (e.g., eman2/libpyEM/processor.pyste). For a function that return a pointer, a return-policy must be defined in the pyste file. The typical cases are:
   (a) If the function returns a pointer allocated in this function, do: set_policy(YOUR_FUNCTION, return_value_policy(manage_new_object))
   (b) If the function returns a static pointer , do: set_policy(YOUR_FUNCTION, return_value_policy(reference_existing_object))
   (c) For other cases, do: set_policy(YOUR_FUNCTION, return_internal_reference())
3. Run script: eman2/libpyEM/create_boost_python

3.12.5 Using FFTW

EMAN2 works with both fftw2 and fftw3. A user makes the choice at compile time. A standard interface is defined to do fft:

```
class EMfft {
public:
    static int real_to_complex_1d(float *real_data, float *complex_data,
                                  int n);
    static int complex_to_real_1d(float *complex_data, float *real_data,
                                  int n);
    static int real_to_complex_nd(float *real_data, float *complex_data,
                                  int nx, int ny, int nz);
    static int complex_to_real_nd(float *complex_data, float *real_data,
                                  int nx, int ny, int nz);
 };
```

3.12.6 Large File I/O

1. **portable_fseek()** should be used for fseek.
2. **portable_ftell()** should be used for ftell.

3.12.7 Euler Angles

- Euler angles are implemented in **Rotation** class.
- ```
  Rotation r = Rotation(alt, az, phi, Rotation::EMAN);
  float alt2 = r.eman_alt();
  float az2 = r.eman_az();
  float phi2 = r.eman_phi();

  float theta = r.mrc_theta();
  float phi = r.mrc_phi();
  float omega = r.mrc_omega();
  ```

3.12.8 Using Numeric Python

- In EMAN2, Numeric array and the corresponding EMData object shares the same memory block.
- Example: Converting EMData object to Numeric numpy array

  ```
  from EMAN2 import *
  e = EMData()
  e.read_image("e.mrc"))
  array = EMNumPy.em2numpy(e)
  ```

- Example: Converting Numerc numpy array to EMData object

  ```
  from EMAN2 import *
  import Numeric
  n= 100
  numbers= range(2*n*n)
  array= Numeric.reshape(Numeric.array(numbers,Numeric.Float32),(2*n,n))
  e= EMData()
  Wrapper.numpy2em(array, e)
  e.write_image("numpy.mrc")
  ```

3.12.9 Using Transformations

Transform defines a transformation, which can be rotation, translation, scale, and their combinations.

Internally a transformation is stored in a 4x3 matrix. $\begin{bmatrix} a & b & c \\ e & f & g \\ i & j & k \\ m & n & o \end{bmatrix}$
  ```

The left-top 3x3 submatrix $\begin{bmatrix} a & b & c \\ e & f & g \\ i & j & k \end{bmatrix}$ provides rotation, scaling and skewing.

Post translation is stored in $\begin{bmatrix} m & n & o \end{bmatrix}$

A separate vector containing the pretranslation, with an implicit column $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ at the end when 4x4 multiplies are required.

The 'center of rotation' is NOT implemented as a separate vector, but as a combination of pre and post translations.

3.12.10 Printing Error/Warning/Debugging Information

- Using the Log Class

  1. In your main() file, set log level: Log::logger() $\longrightarrow$set_log_level(WARNING_LOG);
  2. Log message in different levels: (log functions use the same argument format like printf()).

     LOGERR("out of memory");

     LOGWARN("invalid image size");

     LOGDEBUG("image mean density = %f$\backslash n$", mean);

     LOGVAR("image size = (%d, %d, %d)$\backslash n$", nx, ny, nz);
  3. To log function enter point, use ENTERFUNC; To log function exit point, use EXITFUNC.

3.12.11 Adding Testing Groups

- These group tags are already defined in file, "eman2doc.h":

  - tested0 : code not yet complete
  - tested1 : code complete but untested
  - tested2 : code complete but contains bugs
  - tested3 : tested
  - tested3a : manual testing
  - tested3b : unit test in C++
  - tested3c : unit test in Python
  - tested3d : incorporated into sucessful regression test

- How to use these tag to label testing group:

- add /**@ingroup tested3c*/ to the beginning of a class tested in Python, then the corresponding class will be labeled ”unit test in Python” in doxygen generated document.
- you can also define other grouping tag, just follow the testing group example in ”eman2doc.h”
- a single unit can be labeled for multiple group

3.13 How to Install Boost Python

1. Download ’bjam’ for your platform.

2. Download boost source from http://www.boost.org. Assume the version is boost_1_32_0.

   % cd /usr/local/src

   % tar zxf boost_1_32_0.tar.gz

   % cd boost_1_32_0.

3. Set up environment variables ”PYTHON_ROOT” and ”PYTHON_VERSION”. For example, if your python is at /usr/bin/python then PYTHON_ROOT is ”/usr”. If your python version is 2.2.X, then PYTHON_VERSION is ’2.2’.

   (a) check your shell: % echo $SHELL

   (b) if you are using bash/zsh, do

       % export PYTHON_VERSION=2.2
       % export PYTHON_ROOT=/usr

       if you are using csh/tcsh, do

       % setenv PYTHON_VERSION 2.2
       % setenv PYTHON_ROOT /usr

4. cd libs/python/build

5. run ’bjam’ with your options:

   - linux-x86: % bjam
   - SGI Irix: % bjam ”-sTOOLS=mipspro”

6. login as root

7. % cp -df bin-stage/libboost_python.so* /usr/local/lib

   % cd ../../..; cp -rf boost /usr/local/include

3.14 How to use your own version of python

If the python you want to use in your computer is not found by CMake, you may set up environment variables ”PYTHON_ROOT” and ”PYTHON_VERSION”. For example, if your python is at /usr/local/python2.4/bin/python. PYTHON_ROOT is ”/usr/local/python2.4”. if your python is 2.4.X, PYTHON_VERSION is ’2.4’.

3.15 How to Install Numeric Python

From the website http://sourceforge.net/projects/numpy and download version 22.0 of Numeric Python.

For windows, run the binary installer and the installation is complete. Other users must download the source code and install manually as follows:

- get source code Numeric-XX.Y.tar.gz
- % gunzip Numeric-XX.Y.tar.gz
  % tar xf Numeric-XX.Y.tar
- login as root
- % cd Numeric-XX.Y;
  % python setup.py install

Index