

# The diffmic CVS archive

September 20, 2007

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Philosophy and goals . . . . .	3
1.1.1	Philosophy of CCP4 and Macro-EM . . . . .	4
1.1.2	Suggested guidelines . . . . .	4
<b>2</b>	<b>Using the archive</b>	<b>6</b>
2.1	Accessing the IDL routines . . . . .	6
2.2	Structure of the archive . . . . .	6
2.2.1	Access using the unix command line . . . . .	7
2.2.2	Access using WinCVS . . . . .	8
2.3	Updating the on-line version . . . . .	8
<b>3</b>	<b>Areas suggested for future work</b>	<b>11</b>
3.1	Common file formats for reconstruction . . . . .	11
3.2	Routines for evaluating raw data . . . . .	11
3.3	C/C++ routines for reconstruction on parallel computers . . . . .	11
3.4	Data visualization . . . . .	11
3.5	Stylistic comments . . . . .	12

# List of Figures

2.1	WinCVS checkout . . . . .	8
2.2	WinCVS update . . . . .	9
2.3	WinCVS add . . . . .	10

# Chapter 1

## Introduction

### 1.1 Philosophy and goals

Diffraction microscopy is a coherence-based technique that could not exist without modern computers: iterative algorithms are used to phase the measured diffraction data and produce a real-space image, and computers are also integral to reliable collection of data throughout a tilt sequence.

Various groups in this area of research are in friendly competition with each other, in that we are all exploring different tricks in phase retrieval algorithms and processing procedures that are peculiar to our particular experiments and data. This is assuredly good, because competition stimulates creativity. At the same time, we are all using many of the same tools, including algorithms that involve hundreds or thousands of Fourier transforms, application of common constraints, and similar issues in 3D data assembly and visualization. As the size of our data sets increase beyond the capabilities of standard desktop computers, parallel computational approaches are beginning to be employed including message passing interface (MPI) enabled routines, yet there is relatively little standard code presently available for many of the common operations needed for diffraction imaging data processing.

Just as specialized experimental apparati can be constructed in part by bolting together common subunits such as motorized translation stages and CCD cameras, specialized software can be constructed from a framework that includes available subroutines. When common standards are employed, these subroutines can be written, tested, and documented by researchers worldwide, so that any individual can make greater progress by writing specialized code that exploits a common framework. This approach has been exploited to great advantage in the protein crystallography community, where programs required for tackling different steps of structure determination have been written to accept input and provide output to each other (Sec. 1.1.1); numerous other examples exist, including the development of the Linux/GNU operating system and experimental control systems such as EPICS and SPEC.

Based on a workshop held in 2004<sup>1</sup>, we have begun to develop standards and available software modules for diffraction imaging as described in the document you are now viewing, which is

---

<sup>1</sup><http://xray1.physics.sunysb.edu/~jacobsen/workshop.html>

available either as a [PDF file](#) or as a [web link](#).

### 1.1.1 Philosophy of CCP4 and Macro-EM

The CCP4 project (<http://www.ccp4.ac.uk/>) for protein crystallography provides a good model:

Unlike many other packages, particularly for small molecule crystallography, the CCP4 suite is a set of separate programs which communicate via standard data files, rather than all operations being integrated into one huge program. This has some disadvantages in that it is less easy for programs to make decisions about what operation to do next—though it is seldom a problem in practice—and that the programs are less consistent with each other (although much work has been done to improve this). However the great advantage arising from such loose organisation is that it is very easy to add new programs or to modify existing ones without upsetting other parts of the suite. This reflects the approach successfully taken by Unix. Converting a program to use the standard CCP4 file formats is generally straightforward, and the philosophy of the collection has been to be inclusive, so that several programs may be available to do the same task. The components of the whole system are thus a collection of programs using a standard software library to access standard format files (and a set of examples files and documentation) available for most Unix operating systems (including Linux), as well as Windows and Mac OS X. Programs are mostly written in C/C++ and Fortran 77.

Another source of inspiration is the Macro-EM project (<http://www.macro-em.org/>):

Our goal is to develop computational technology that will make it possible to get high-resolution density maps, and to do so from EM images of large, isolated macromolecular particles at a high rate of throughput. We want to develop versions of single-particle software that will take full advantage of modern, affordable, and most importantly highly parallel machine architectures. The parallel machines are able to process large amounts of data simultaneously, thus completing the computing tasks required for cryo-EM in a realistic amount of time... Our strategy in developing optimized software for processing large amounts of data in a relatively short time, giving high resolution, is to first implement pilot versions of desired code on multiprocessor clusters that are based on commodity PC hardware. The ultimate goal is to develop the computational technology that will improve both resolution and throughput when calculations are run on machines that are affordable (a) for individual laboratories, (b) as shared instrumentation, or (c) as dedicated machines, run for community as multi-user facilities.

### 1.1.2 Suggested guidelines

Here are the present guidelines for writing code:

- *Code should be transportable across machines small and big.* We would like the same source code to serve for both single-processor Linux PCs and for clusters of Apple Xserve G5 machines, to pick an example. As an example, we might choose to have a wrapper subroutine for FFT routines that decides at compile or even run time to call the FFTW routine if on a single-processor computer, or the Apple `dist_fft` library if on an Apple cluster.

This also means we should try to use languages that are widely available and understood. Basic C++ code fits this goal pretty well.

- *We should use fairly long and descriptive names for routines and variables.* It's easy to understand at a glance what's going on with a routine like

```
dm_array_math.apply_support_constraint( complexarr array, supportarr
support, float beta).
```

- *We should identify authorship of routines, and document changes, in the .cpp files.* For example, we might have a class `dm_array_math` which might have one routine written by person A, and another by person B. Each routine within this class should have something in it like

```
// 15-aug-2005: original program written by Daffy Duck.
// 16-aug-2005: modified by Tweety Bird to handle both single
//    and double precision.
```

- *Data files should be transportable across machines.* We will want all our files to have a byte at the beginning that says if they are in big-endian or little-endian order; it would likely be good to have this match the byte order on the machine that's doing the most work.

It's also useful to have a byte that records the file version so that if additional information needs to be added later on

- *Heavy lifting should be done by compiled code.* IDL is nice for examining and evaluating data files, but iterative calculations, and assembly of multiple 2D files into a 3D diffraction data set, represent jobs that are best left to compiled code that can be run on a cluster machine if available.
- *Self-documenting scripts are nice.* Driving calculations by using some kind of script language text file means that you can document how a particular calculation was done.

# Chapter 2

## Using the archive

### 2.1 Accessing the IDL routines

This archive includes both C/C++ code and IDL program code. To make sure your IDL session knows about the `diffmic` archive routines, you need to know where they are located and tell IDL about this information by setting a `IDL_PATH` preference. archive into your

**Unix or Mac, IDL 6.2 or later:** Let's assume that you have done a CVS checkout of `diffmic` from your top-level directory. Run IDL, and type the following:

```
pref_set, 'IDL_PATH', '<IDL_DEFAULT>:+$HOME/diffmic/idl', /commit
```

Exit IDL, and then you should be set for the future.

**Windows, IDL 6.2 or later:** Let's assume you have put the CVS checkout of `diffmic` into the directory `c:\cvs\diffmic`. Run IDL, and type the following:

```
pref_set, 'IDL_PATH', '<IDL_DEFAULT>;+c:\cvs\diffmic', /commit
```

Exit IDL, and then you should be set for the future.

**Unix or Mac, IDL 6.1 or earlier:** Let's assume that you have done a CVS checkout of `diffmic` from your top-level directory. Put the following command into your `.bashrc` file to let IDL find the code:

```
export IDL_PATH="<IDL_DEFAULT>:+$HOME/diffmic/idl"
```

**Windows, IDL 6.1 or earlier:** On a Windows machine, you specify this within the IDL developer environment. Go to *File*→*Preferences* and pick the *Path* tab. If you have, for example, checked out the `diffmic` archive to a directory `c:\cvs\diffmic\idl`, you will want to click on *Insert* and add that directory along with a preceding check to search subdirectories as well.

### 2.2 Structure of the archive

The structure of the `diffmic` archive is as follows:

**c:** C/C++ source code. This top-level directory includes `datatype.h` for defining variable types of specific size, and the `log_err` class for error reporting.

**diffmap:** Enju Lima's code for demonstrating the difference map algorithm.

**dist\_fft:** the Apple FFT routines.

**expt:** code for running the experimental apparatus at ALS beamline 9.0.1.

**hawk:** Philippe Maia's reconstruction algorithm.

**r3d\_mpi:** Anton Barty's 3D reconstruction software

**retriever:** Pierre Thibault's 2D difference map program

**test:** programs used for testing things.

**util:** utility support routines like the `dm_fileio` class.

**doc:** includes this document which is `diffmic.tex`, available on-line as either a [PDF file](#) or as a [web link](#).

**recon:** code for reconstructing diffraction data. Further details are given in the file `diffmic/doc/recon/diffmic_recon.tex`, which is used to produce on-line documentation of either a [PDF file](#) or a [web link](#).

**expt:** documentation on the apparatus at ALS beamline 9.0.1. The main document is the file `diffmic/doc/expt/diffmic_expt.tex`, which is used to produce on-line documentation of either a [PDF file](#) or a [web link](#).

**idl:** source code for IDL from Research Systems Inc. This top-level directory includes routines like `dt_ipar.pro`, `read_dt.pro`, and `write_dt.pro`. See Sec. 2.1 for a description of how to make it accessible to your IDL programs. These are some of the subdirectories:

**expt:** code used for running the apparatus at ALS beamline 9.0.1.

**fakecell:** code used for building a fake 3D object.

**merger:** Henry Chapman's program for merging multiple 2D exposures.

**test:** test routines.

**util:** routines like `dm_read_adi.pro`.

### 2.2.1 Access using the unix command line

This applies to Linux, Cygwin in Windows, the Mac OS X terminal window... To access the archive, you must first do the following:

```
export CVSROOT=":ext:diffmic@xray1.physics.sunysb.edu:/home/diffmic"
```

You can now get the archive for the first time by typing `cvs checkout diffmic`. From then on you can go into the `diffmic` directory and do `cvs update` followed by a `cvs commit` if you have changes to send back to the archive. Beyond that, see the web page for CVS (<http://www.cvshome.org>) for more info.

## 2.2.2 Access using WinCVS

Examples of using WinCVS (<http://www.wincvs.org>) to access the archive are shown in Figs. 2.1, 2.2, and 2.3.

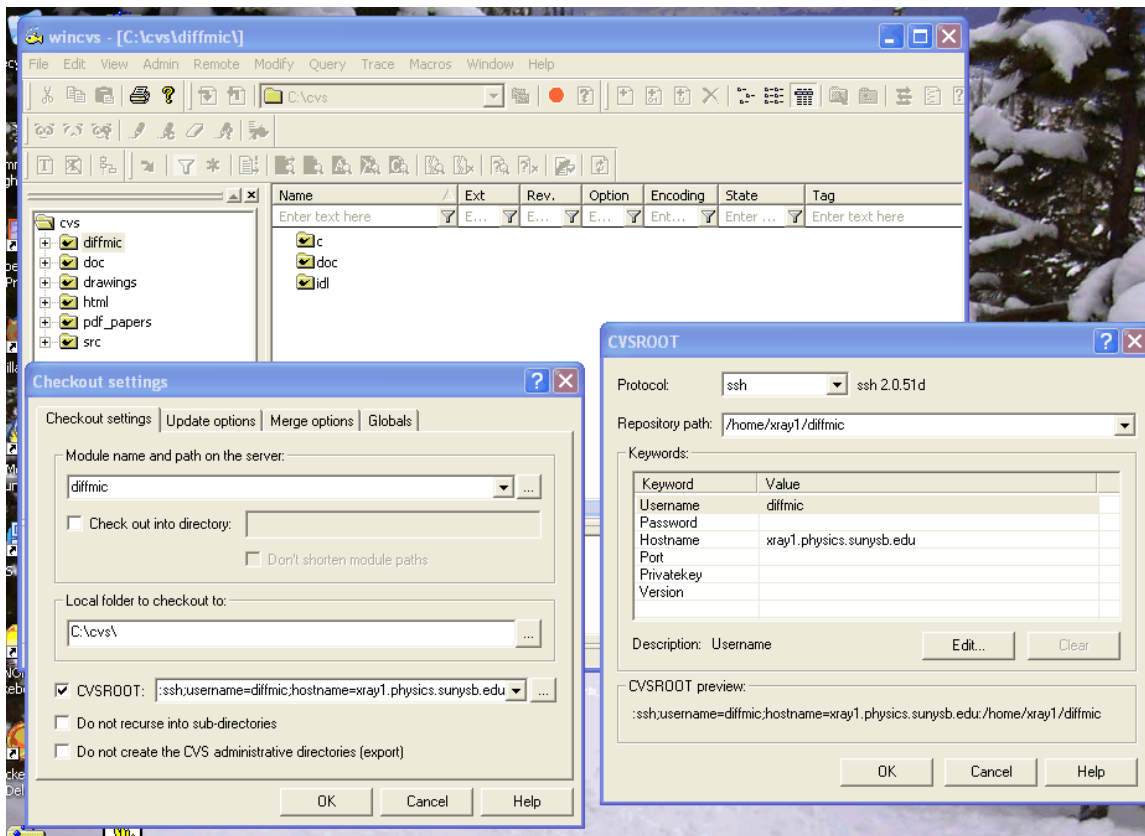


Figure 2.1: A screen snapshot of using the WinCVS *checkout* procedure to set up access to the *diffmic* archive. WinCVS version 2.0 was used in this example. In WinCVS you first select *Remote* → *Checkout module*, which gives you the window at lower left. Within that window, you can click on the “...” to the right of *CVSROOT* to get the screen at lower right. In that screen, you set *Protocol*, *Repository path*, and edit the *Username* and *Hostname* as shown. You should then be able to check out the *diffmic* repository.

## 2.3 Updating the on-line version

Log on as *micros* on *xray1.physics.sunysb.edu*. Do `cd /cvs/diffmic` and do a CVS update. Do `cd /cvs/diffmic/doc` and do `make` to rebuild this document on the web, where it appears as either a [PDF file](#) or as a [web link](#).

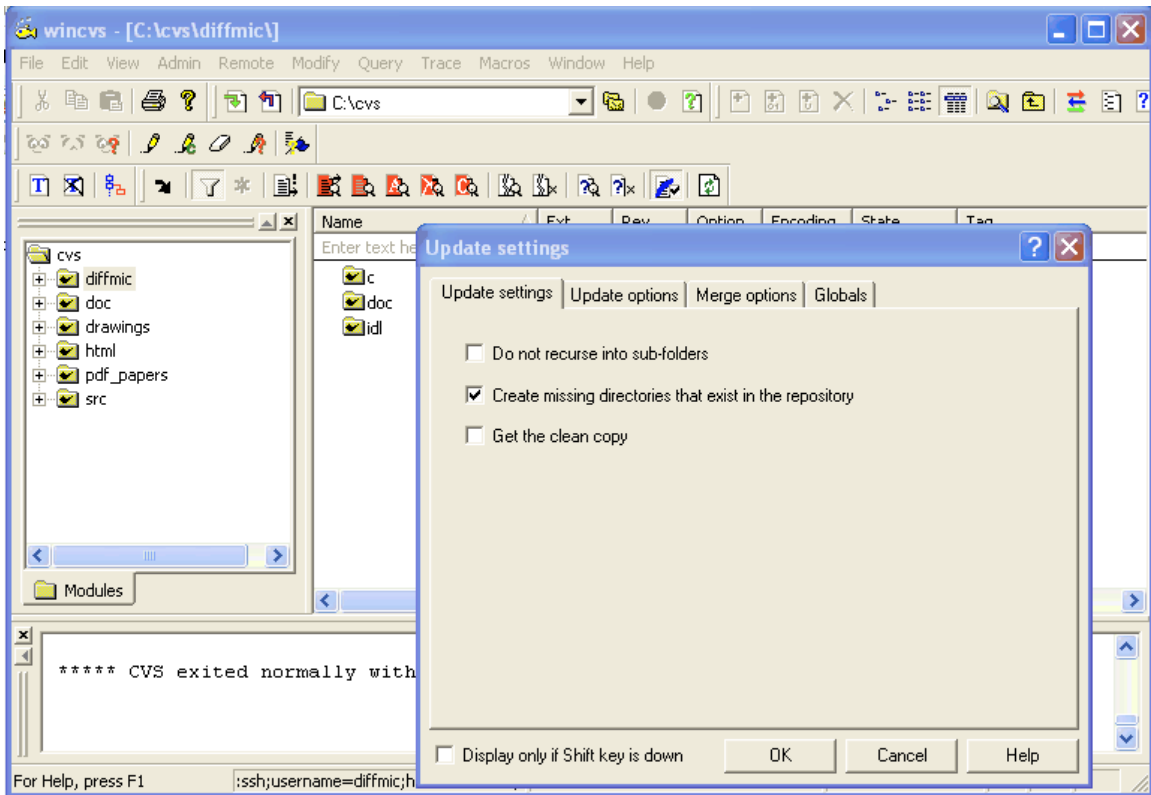


Figure 2.2: A screen snapshot of the WinCVS *update* procedure.

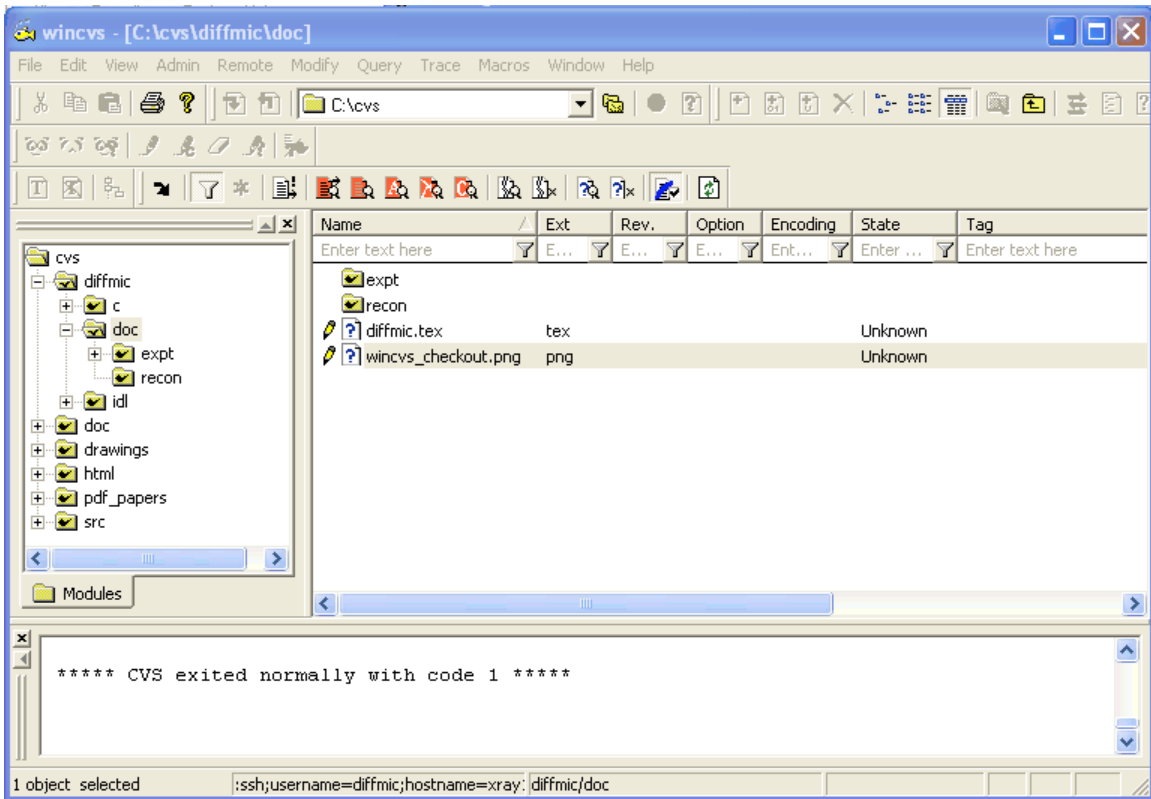


Figure 2.3: A screen snapshot of the WinCVS *add* procedure.

# Chapter 3

## Areas suggested for future work

### 3.1 Common file formats for reconstruction

Use big-endian binary format. Provide read-write routines in C, IDL. Assembled diffraction intensity: common format for 2D reconstructions from `merger`, 3D reconstructions from `xewald`. Include provision for specifying the experimental error of each pixel/voxel. Support mask: define area of support as binary and possibly graduated bitmapped image Iterate amplitude: present result for real or complex object in real space

### 3.2 Routines for evaluating raw data

Finding the location of the zero-frequency center of a diffraction pattern. Characterizing background noise and information content. Displaying the autocorrelation of different diffraction sub-regions (S. Marchesini). Isolating the beamstop, as well as saturated pixels. Estimating the support from the autocorrelation.

### 3.3 C/C++ routines for reconstruction on parallel computers

Message Passing Interface (MPI) enabled code on multiple nodes can be very effective for reconstructing large data sets. See poster by A. Barty. Python scripting of C/C++ routines? 2D and 3D Fourier transforms, such as the Apple `dist_fft` library. Routines for applying support, Fourier modulus constraints (HIO, difference map, shrink-wrap). Routines for file input/output. Routines for averaging iterates, and evaluating reconstructions. Refinement of reconstructions: improved estimates of true tilt, scaling, centering of 2D diffraction data?

### 3.4 Data visualization

Routines for interactive display of 2D and 3D reconstruction. Routines for generating movies of rotating 3D reconstructions with partial transparency

## 3.5 Stylistic comments

Imagine Dana Carvey’s “church lady” wagging a finger at you while s/he says things in a scolding voice:

- Be careful to always do `cvs update` before `cvs commit`!
- Whenever possible limit lines to 80 characters. Maybe your favorite editor does automatic line wrap-around, but it’s nicer for all concerned to do this manually.
- Please preserve the tabbed-indentation of source code files. If it’s messed up, you can always select the entire file in a buffer and do the Emacs command `M-x indent-region` to restore the correct tabbing for source code files.
- Please be careful about using your local system’s convention for end-of-line/carriage-return stuff, so that files don’t get confused at the end of each line.