CCP4 NEWSLETTER ON PROTEIN CRYSTALLOGRAPHY

An informal Newsletter associated with the BBSRC Collaborative Computational ProjectNo. 4 on Protein Crystallography.

Number 46 **Summer 2007**

Contents

Editor: Francois Remacle Daresbury Laboratory, Daresbury, Warrington, WA4 4AD, UK

Mosflm 7.0.1 and its new interface - iMosflm 0.5.3

Harry Powell, Andrew Leslie and Geoff Battye
MRC Laboratory of Molecular Biology, Hills Road, Cambridge CB2 2QH
Email: harry@mrc-lmb.cam.ac.uk, andrew@mrc-lmb.cam.ac.uk

Introduction

CCP4 provided the funding to allow a new graphical user interface (GUI) to be developed for the diffraction image integration program Mosflm. The result, known as iMosflm, is a new interface that has a straightforward and logical flow through processing diffraction data. It has been designed so that a novice user can easily process datasets and it also has the functionality to allow more advanced users to deal with difficult data. The feedback to the user has been tailored to provide the most useful items in as clear a way as possible so that the various processing parameters can be optimized. Output which was only available in Mosflm in tables is now presented by iMosflm in the most appropriate form, e.g. pie-charts, histograms and line graphs. Variables being refined during processing are displayed as real-time graphs, which allow the user to see immediately if any problems arise.

iMosflm is written in TclTk, and can be installed and run on most modern computing platforms, including Linux, OS X and Microsoft Windows.

Underlying changes have been implemented in Mosflm to facilitate the development of iMosflm, and which also make processing more robust and less likely to cause the user problems. Mosflm itself can be run from the command-line (e.g. in a UNIX shell window or a MS-Windows command prompt), so it can be used in shell scripts, and the traditional X11 GUI is still available for all platforms for which it was previously available.

The vast majority of the work on iMosflm was carried out by Geoff Battye between 2003 and 2006; this culminated in the release of iMosflm 0.4.5 in August 2006, which required the installation of a beta-release copy of Mosflm version 6.2.6. The latest version (described here, v0.5.3), is the first to run with a full release copy of Mosflm (v7.0.1).

Both iMosflm and Mosflm can be obtained by following the links at http://www.mrc-lmb.cam.ac.uk/harry. A normal pre-requisite is that the CCP4 suite should be set up correctly on the target platform.

Background

Mosflm is an integration program with a long history, with its origins dating back to the 1970's. A GUI was first suggested in 1992, and this was implemented using John Campbell's "xdl_view" X-windows package shortly after. While this represented a great leap forward in usability, by the end of that decade it was beginning to look rather old-fashioned and its fairly rigid format was starting to restrict development; its use also prevented porting Mosflm straightforwardly to MS-Windows. The decision was made to develop a new, more portable, interface using modern object-oriented design principles; at the time, the scripting language TclTk with the [incr]Tcl and [incr]Tk extensions was well-

established and provided the required functionality. Object-oriented programming (OOP) is the modern standard method in software development for a variety of reasons, including straightforward re-use of code and modular implementation of functionality.

There are two main ways of interfacing a GUI with another program; communication can be carried out by parsing output in log files, or a more intimate connection can be made by using TCP/IP sockets and applying a consistent structure to the information passed between the components. A ubiquitous example of the latter is the use of encapsulating information in HTML to pass from a web server to a browser. We chose to use sockets for linking iMosflm/Mosflm, and developed a local XML (eXtensible Markup Language) for encoding information to be sent from Mosflm to iMosflm; all communication in the other direction is in the form of standard Mosflm commands. The dictionary of available commands was extended to allow for this new method of running Mosflm; a consequence of this has been that Mosflm itself has become more flexible in its command-line scripting.

Installation of iMosflm 0.5.3

Full installation instructions are on the website.

The following components should be installed on your system;

- Current CCP4 distribution
- TclTk 8.4
- Mosflm version 7.0.1
- iMosflm source

Since the normal CCP4 installation process now gives the option of installing the ActiveTcl TclTk package, this does not need to be repeated for iMosflm. A copy of TclTk built for Tru64 UNIX is available on the iMosflm download site.

Mosflm 7.0.1 can be downloaded as a pre-built binary for the available platforms, or built from the Mosflm source code by the user.

iMosflm 0.5.3 is distributed as a tar'd and gzip'd archive containing over 200 files; these are mostly ASCII source files and no compilation or other building is necessary. The MS-Windows version is distributed as a zip file, containing the iMosflm source as well as a Mosflm 7.0.1 executable and a Tcl script file for launching the interface.

Installation should be straightforward. The archive should be extracted using

- `tar -xzf imosflm.tgz` (UNIX-based systems)
- double-click on the "imosflm.zip" icon (MS-Windows) and follow the instructions.

iMosflm 0.5.3 runs on Linux, Mac OS X, Tru64 UNIX and MS-Windows. There are problems with porting some of the TclTk components to Irix, so it is unlikely that iMosflm will run on SGI MIPS workstations.

UNIX-based systems

You need to set up the environment variables MOSFLM_WISH (to point to the "wish" you are using) and MOSFLM_EXEC (to point to the Mosflm executable).

Run the program by typing the full path to the script (or put it in your path) "imosflm/src/imosflm"; this file contains a Bourne shell script to set up variables and perform a number of checks before trying to start the interface itself.

A new feature in version 0.5.3 is the addition of command-line parameters; start imosflm with the option "--help" for a full list, which will be added to in the future.

MS-Windows

Double-clicking on the script "imosflm.tcl" in the top-level "imosflm" folder takes care of setting up the various environment variables for you (do not confuse this with the file of the same name in the folder "imosflm\src"), and starts iMosflm itself.

Running imosflm

On any platform, you should be rewarded by the appearance of an interface with a large, mostly blank pane; this has a number of large buttons down the left-hand side (see Figure 1), and a number of options across the top. The best source of information on how to run the program is the tutorial available online in the iMosflm documentation at http://www.mrc-lmb.cam.ac.uk/harry, but it is worthwhile going through the options briefly here, to give an idea of how to run the interface.

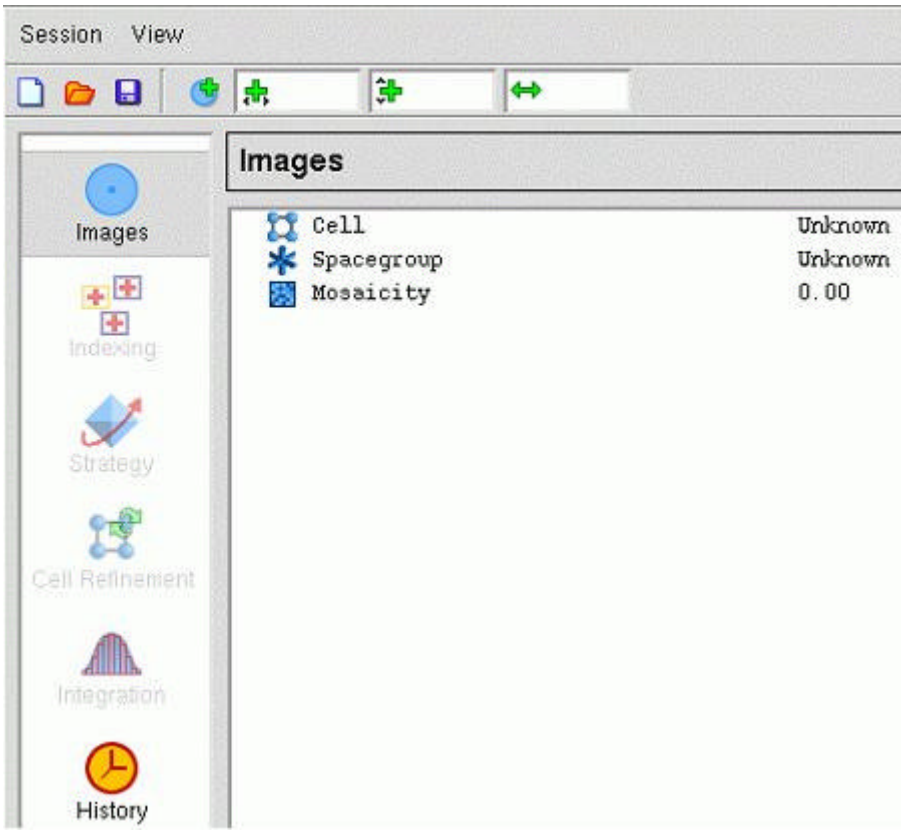


Figure 1. Appearance of the iMosflm GUI on starting the program.

Any options not available at the current stage of processing are greyed out, and cannot be selected. At the start of the process, therefore, the only options available from the list down the side of the GUI are to choose "Images" or to examine the "History". The other options, i.e. "Indexing", "Strategy", "Cell Refinement" and "Integration" cannot be done before

images have been selected so these are greyed out. Clicking the mouse on either "Session" or "View" (both on the top line) gives a drop-down menu with further options. The four buttons immediately below are (reading from left to right) to start a new session, read an existing session or save the current session (in a file which can be read in later) and to read in an image list for processing.

Once an image list has been read, the "Indexing" option becomes live since this is now a possible action. In addition, an image display window opens which displays the first image in the list chosen (Figure 2). This has a number of options to enhance the display in various ways, each of which can be chosen from the menu bar. Note that "tooltips" appear which describe the action of the buttons if the mouse cursor is left over them for a few seconds.

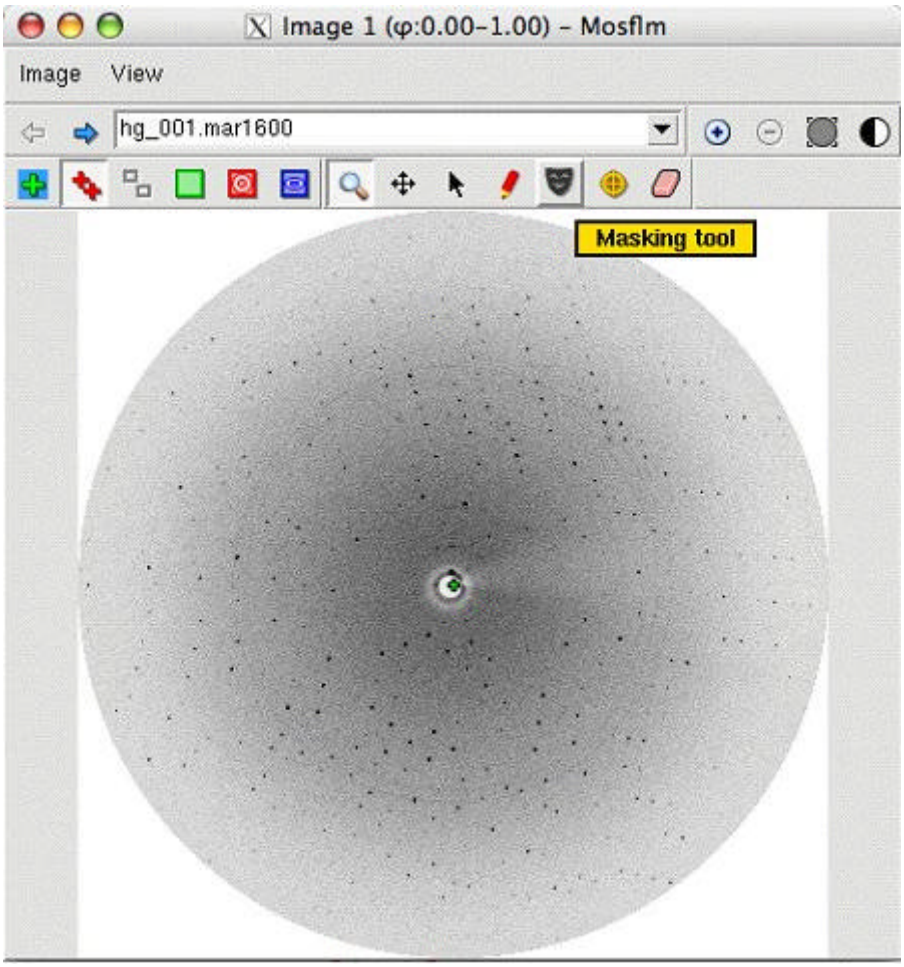


Figure 2. The iMosflm image display window.

Following indexing, all actions are possible, so the "Strategy", "Cell Refinement" and "Integration" buttons become live. As indicated above, the best place for further details on running the interface is the tutorial.

Changes in Mosflm (ipmosflm) 7.0.1

The major change over previous versions (6.2.6 and earlier) is that the autoindexing code has been enhanced considerably to make it more robust; it can now index images of much lower quality than was possible previously in an entirely automatic way. In addition, the detector parameter refinement has been improved to make it more stable when processing weak images (it should no longer be necessary to "fix" parameters like "tilt" and "twist").

New detectors supported by Mosflm are the ADSC Q270 and the Dectris Pilatus pixel array detector (support for this is not yet included in iMosflm).

Other changes are enhancements to communicate with iMosflm, the DNA automation project and many small bug fixes. The change in major version number indicates that this is the first Mosflm version which deals properly with iMosflm.

The bugs that have been addressed mean that Mosflm will prove to be an even more robust program to process diffraction images. With the exception of the MS-Windows version, all Mosflm executables include the traditional X11 GUI that uses the xdl_view libraries.

Mosflm will now build and run on MS-Windows, using either Visual-C and Visual Fortran or the Mingw gcc/g77 pairing; indeed, the MS-Windows executable distributed with iMosflm is built on a Macintosh running OS X and using a cross-compiler version of gcc/g77.

Acknowledgements

In particular, we would like to thank François Remacle for his help in preparing the MS-Windows port using Visual C and Fortran. Also, we would like to thank all the users who have provided us with bug reports and suggestions for improvements and who have tested the software so thoroughly.

Latest developments in Diffraction Image Library

Francois Remacle, Graeme Winter, STFC Daresbury Laboratory Warrington WA4 4AD, United Kingdom

Introduction

This library provides a single way of handling diffraction image that can be originally of various different formats. It has been designed so that you only need to use a single object whatever the format of your image is. All the work of identifying what type of image is done internally.

Currently, the following formats are supported by this library:

- ADSC
- MAR
- RIGAKU
- CBF
- BRUKER

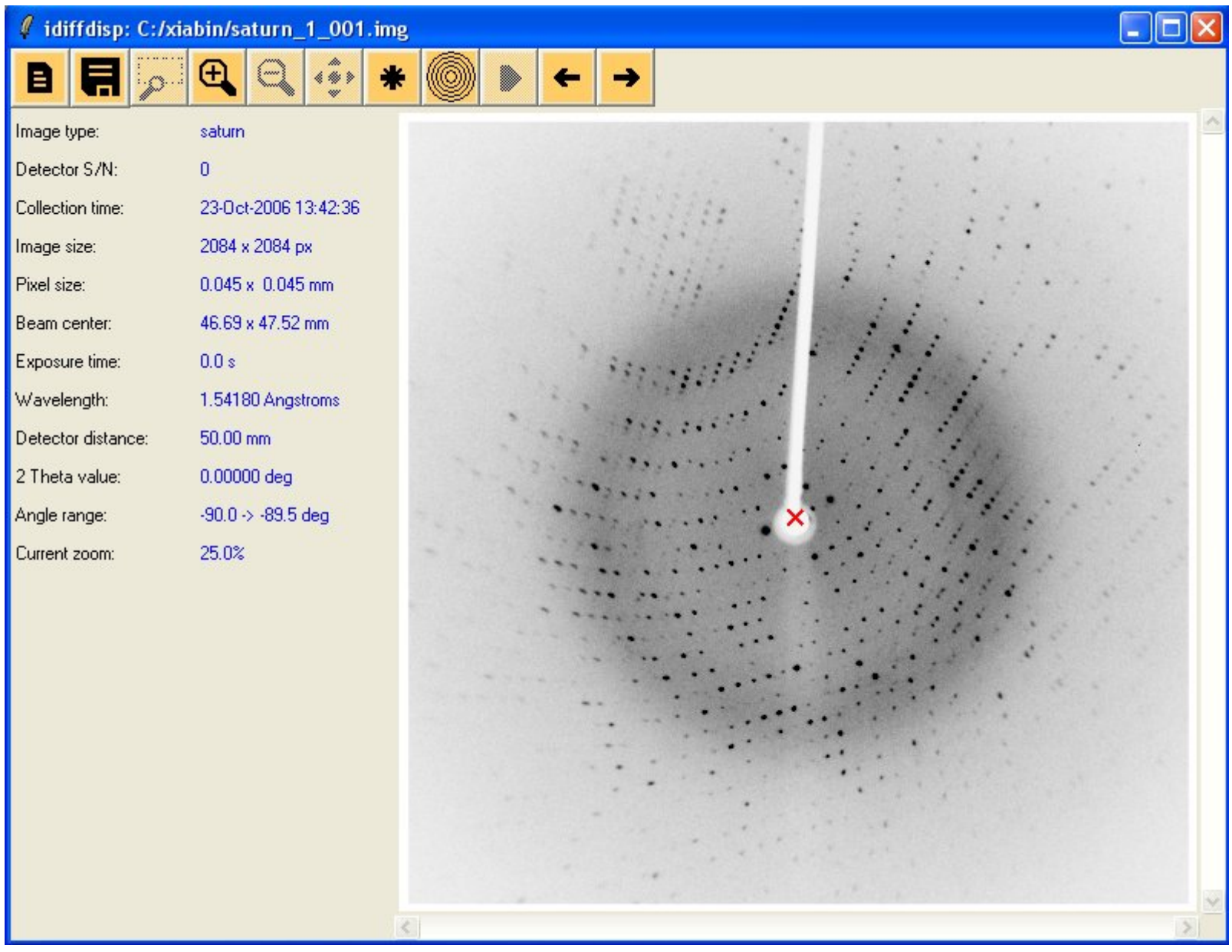
The library also contains a PeakList object that can be populated with the peaks (spots) found on particular images and a set of related operations.

The Diffraction Image Library comes with Tcl and Python interfaces so that the library can be used with both of these scripting languages as well. A previous article was published in the previous newsletter you can see it by clicking here

Alternatively you can come and see me at the BSR conference in manchester where I will be presenting a poster on the Diffraction Image Library.

idiffdisp (CCP4i diffraction image display)

As a first application to use Diffraction Image library, this image viewer will be incorporated in release 6.1 of ccp4. Below is a screenshot of what the main window looks like after opening a single image. The name idiffdisp might change in the future.



As you can see it is divided in 3 parts: The toolbar, the information panel on the left, originally containing the header information and the zoom level and the image on the right. Let's quickly explain the toolbar.

Open menu: Let you open a single image, a maximal image from a sector or make a movie sequence out of a sector. Also give you the possibility to reopen recently opened image/sectors.

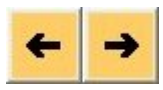
Save menu: Let you save a single image or a maximal image to jpeg format or save a movie sequence as a gif file. This require image magick convert program to be installed.

Zoom in and out: from a range of zoom, currently 10%, 25%, 33%, 50%, 66%, 100%, 200%.

spots finding: Show/Clear spots found on the image, the original I/sigma value is 2.0 but as soon as this option is on. A slider will appear on the left panel to let you adjust the value as shown with the maximal image screenshot below.

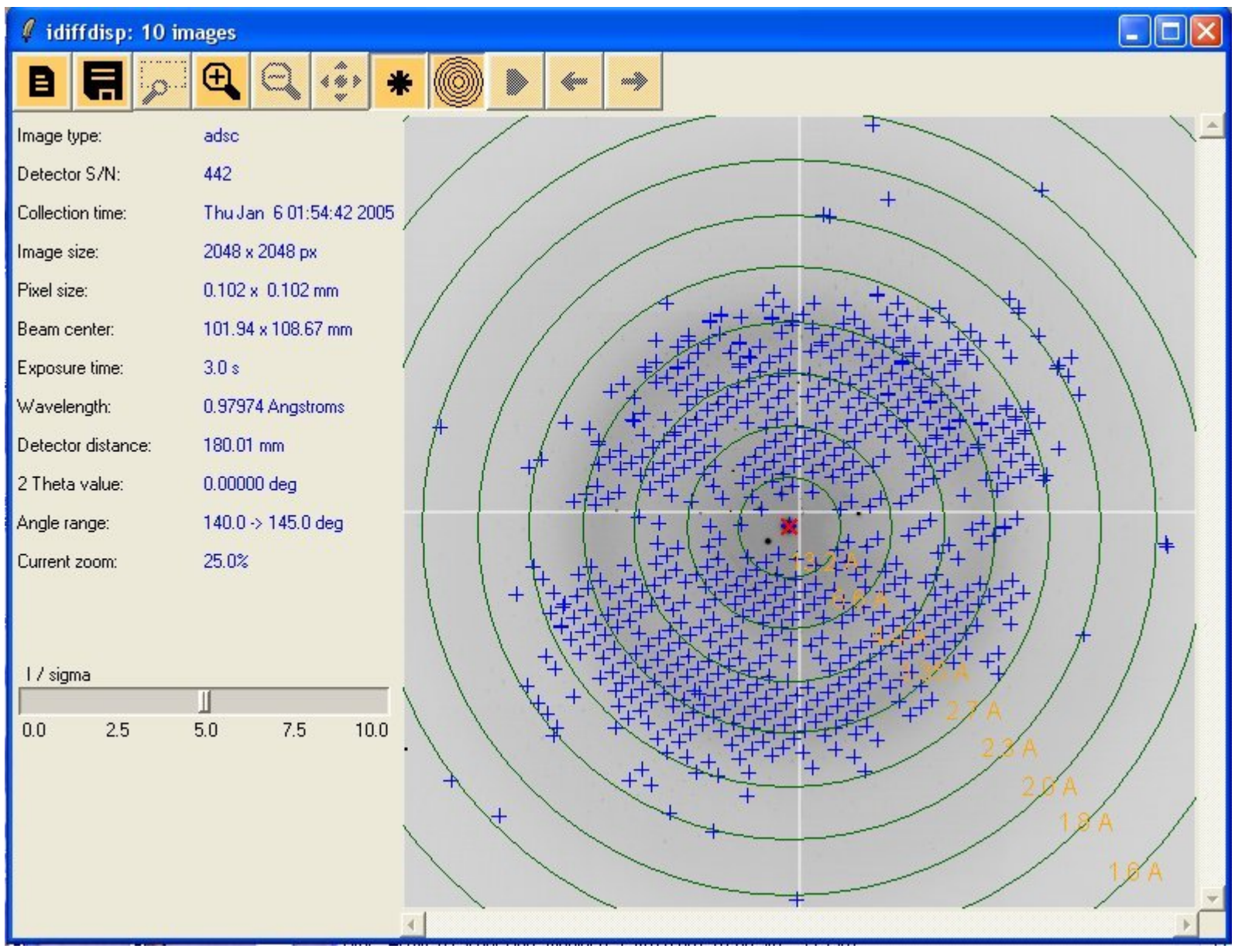
Resolution circles: Show/clear resolution circles calculated for the image of maximal image.

Play: Play the movie sequence, only available in movie mode. Note that in movie mode similarly to the I/sigma slider. Two slider appears on the left panel. The first one let you specified the inter frame delay and the second let you slide through your frames. Further to these slider a checkbox appear to let you choose if you want the movie to rollback or not.



Previous/Next: In image mode, these will open the previous/next image in directory. In movie sequence mode this will display the previous/next frame.

There are a couple of button that are present but not active yet as the function behind them are still being developed. Below you can see an example of a maximal image with spots and resolution circles



Finally, here is an example of a saved movie sequence done with the option save as gif.

Automask

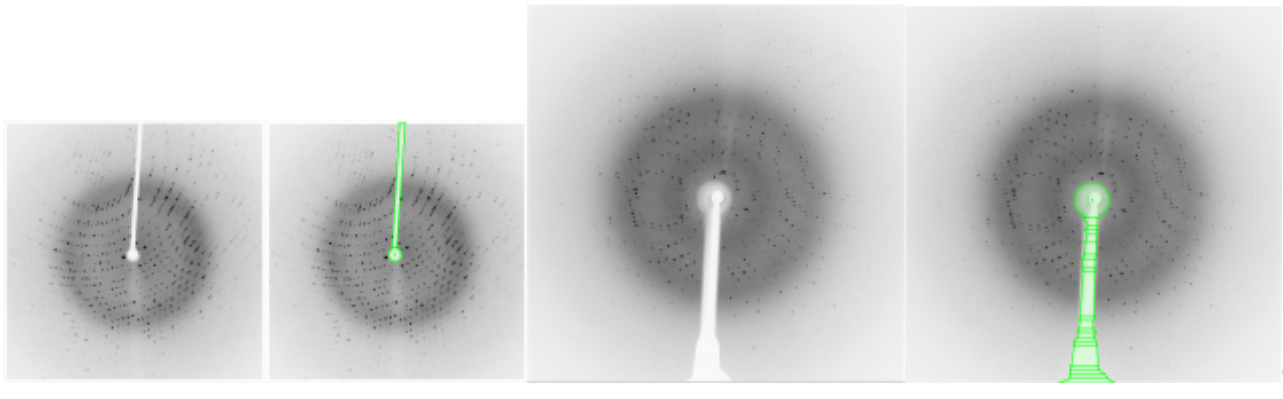
This is a new function that is still being developed but that already provides very promising results in automatically generating a mask for the backstop area and the arm of the backstop (if present on the image). Here is a quick description of the way it works.

The first step is to try to determine the backstop area. To do so we shoot rays in 3 different directions 120 degrees apart from each other. We decide to stop as soon as the standard deviation becomes "too big" in comparison with the average pixel count along the rays. With the 3 points obtained we fit a circle passing through them. We repeat the process 10 times changing the initial direction. We then take the average centre coordinates and the largest radius to be the characteristics of our backstop area. We then try to modify it a bit by try to move/grow the circle around to see if there is any direction in which we would lower the standard deviation of the whole disc, and stopping to grow if the standard deviation gets too big.

The second step is to determine in which direction the arm of the backstop is situated, and mainly if there is an arm at all. To do so, we shoot 360 degrees rays from centre and as before we stop if the standard deviation of the pixel count gets too big compared to its average value. If by chance we reach the edge of the picture before that than we consider that there is an arm and we use that direction to determine it. If after doing a 360 degree search we did not find a direction where we reached the edge. We try to refine the best result (i.e the direction where we when the farest from the centre) by starting not from the centre of the backstop circle but from its edge. We then restart the search around the best direction so far. If after refining we still did not find any direction we consider there is no arm to the backstop.

The third step happens when we successfully detected an arm direction. We firstly try to do a simple fitting. To do so, starting from the edge of the backstop circle in the direction found, we shoot two rays parallel to the edge of the picture we touched, one in each direction (i.e left and right or up and down depending on which edge of the picture we stopped). We then shoot another two rays from the edge of the picture. With the four points found we can fit a quad to cover the arm. To see if we consider the quad acceptable we evaluate the average and standard deviation of the pixel count within the quad. If it is not too big we consider we have our solution. If it is too big we consider that the shape of the arm might be tricky and we perform a complex fitting. The complex fitting works the same way as the simple fitting does, but instead of doing the ray shooting at both end we do it 40 times along the direction of the arm. We try to reduce the number of quads by looking at the angles between their edges. If their are flat enough we merge then into one.

Currently the algorithm is being developed on single image, but in the end it is likely to take the form a command line application that will use from 1 to n images, calculate the minimal image and perform the mask search on them to avoid interferences when stops are nearby the mask edges. Below are an example of a single fit and a complex fit using single images.



Click on the pictures to enlarge them

Acknowledgments

- Rigaku and BRUKER for supplying example code for their format.
- Many different suppliers of example images of various detectors.
- Paul J. Ellis and Herbert J. Bernstein authors of the CBF library on which this library relies to support CBF format.

POINTLESS: a program to find symmetry from unmerged intensities

Phil Evans, MRC Laboratory of Molecular Biology, Hills Road, Cambridge
pre@mrc-lmb.cam.ac.uk

POINTLESS reads one or more files containing unmerged intensities, and prepares them for scaling (SCALA) in various ways. It accepts files produced by MOSFLM or COMBAT in MTZ format, and can also read unmerged files from XDS (XDS_ASCII) and unmerged SCALEPACK output.

Its principal functions are:-

- determining the Laue (Patterson) group from the observed symmetry, and making a guess at the space group from systematic absences.
- in cases where there are alternative indexing schemes, making the indexing consistent with a reference dataset.

If multiple files are input, it will force the batch numbers to be unique (a long-standing irritation), and ensure consistent indexing in cases where there are alternatives. Dataset names may be edited. It can thus be used to prepare files for input to Scala, as a simpler alternative to SORTMTZ.

An early version of the program was described in Evans (2006), but more recent versions have additional options and some different scoring schemes. The program uses the clever facilities of Ralf Grosse-Kunstleve's cctbx library (Grosse-Kunstleve *etal.*2002) for its symmetry handling, as well as Clipper (Cowtan 2003) and CCP4libraries.

Determination of Laue group and space group

The true spacegroup cannot be definitely known until the structure has been solved and satisfactorily refined, since it is easy to be misled by pseudo-symmetry or twinning. When the diffraction images are indexed, a lattice & crystal class is chosen based only on the cell dimensions, which is not a reliable guide to the true symmetry (it is perfectly possible for example to have a orthorhombic cell with *a* nearly equal *b*). However, by examining how well intensities which are related by potential symmetry agree with each other, we can make an estimate of the likelihood of possible Laue groups. The Laue group has the symmetry of the diffraction pattern, and is the rotational par tof the space group plus a centre of symmetry (Friedel symmetry) and any lattice centering (this is essentially the same as the Patterson group). For chiral space groups, ie for all macromolecules, there is only one point group corresponding to each Laue group (POINTLESS is general for all space groups, including non-chiral and centrosymmetric ones, but by default will choose only chiral groups.) The hierarchy of symmetry determination is: crystal class(lattice symmetry); Laue group (diffraction symmetry); point group; space group(including translational symmetry such as screw axes).

Normalisation. In order to do any scoring of agreement between intensities, we need to put the reflections at least approximately on the same scale. Proper scaling requires knowledge of the symmetry, which is what we are trying to determine, but we can do a rough job by normalising intensities to E^2, by making $\langle E^2 \rangle = 1$ over all resolution ranges. This done by fitting a B-factor to resolution bin averages, then smoothing the residual deviations from $\langle E^2 \rangle = 1$for each resolution-bin with a spline function. Since radiation damage generally increases the apparent B-factor, a simple-minded correction is made by fitting a B-factor as a linear function of "time" (at present using the batch or image number as a proxy for time): this makes a substantial improvement in the scores in cases with marked radiation damage. The scale& B-factors are determined separately for each "run" (ie sweep of contiguous images.)

Scoring functions. The main scoring function used in matching intensities is the correlation coefficient (CC), since it is relatively insensitive to the unknown scale. Probability estimates are then derived from the correlation coefficients. Multiplicity-weighted R-factors are calculated, but these are sensitive to the unknown scales, so are not used in ranking. Correlation coefficients do assume that the data all arise from the same distribution, which is why the raw intensities need to be normalised to E^2, otherwise a correlation will be observed just from the variation of $\langle I \rangle$ in particular with resolution. This can be seen by noting that the correlation coefficient between eg x_i & y_i is just the least-squares slope of plot o fall the (x_i, y_i) points: since $\langle I \rangle$ is larger at low than at high resolution, if we plot pairs of potential but not symmetry-related intensities, which are necessarily at the same resolution, then we will see an apparent correlation due to eg a strong low resolution intensity matching another intensity which is likely to be strong, and so on.

Symmetry elements. The highest possible lattice symmetry compatible with the cell dimensions, within quite generous limits (by default 2° and the equivalent on lengths), is chosen as the test symmetry, ignoring the symmetry in the input file. Each rotation axis in the lattice group is scored separately using all pairs of observations related by that rotation. A probability of the axis being present is estimated from the correlation coefficient, using an error estimate σ (CC) derived from the distribution of correlation coefficients between unrelated pairs, proportional to 1/v(Npairs): this allows for a larger uncertainty if we only have a small sample of reflections. The distribution is modelled as a Lorentzian function, centred on an expectation or "ideal" value estimated as an average of (i) the CC for the identity or Friedel operator and (ii) an estimate of $\boldsymbol{E}$(CC) allowing for the observed error estimates($\boldsymbol{E}(CC) = Var(E^2)/(Var(E^2) + \langle \sigma^2(E^2) \rangle$)(Read, personal communication). To allow for the possibility of pseudo-symmetry, the expected value of CC if the symmetry is not present is not assumed to be 0, but is modelled as a declining probability from P(CC|no symmetry) = 1 at CC=0 to P(CC) = 0 at CC = $\boldsymbol{E}$ (CC | symmetry present) and integrated out. Normalising this probability, P(symmetry | CC) = P(CC|symmetry)/[P(CC|symmetry)+ P(CC|no symmetry)] gives a reasonably robust scoring of the likelihood of each possible symmetry element, without too much danger of over-confidence from an accidental high score with a very few observations. This means that it is often possible to get a reasonable estimate of the Laue group even from a small wedge of data.

Laue groups. The list of possible Laue groups, sub-groups of the lattice group down to the minimum P-1, can be generated from all possible pairs of symmetry elements, including the identity. An estimate of the likelihood of each group is calculated using the combination of the probabilities of each symmetry element which is either present or absent in each sub-group. In each sub-group, each potential symmetry element *i* is either present e_i = true, or absent e_i = false, and we have a measured CC_i and $P(CC_i \mid e_i)$ for e_i= true or false. Then for each group, $P(CC \mid group) = \Pi_i P(CC_i \mid e_i)$. This probability is used to rank the possible

Laue groups. Various other scores are also listed for each sub-group: correlation coefficient, a "net Z-score" from the CCs, R-factors and a measure of the lattice distortion from the original unit cell (in cases where the test lattice is higher symmetry than the original assignment). If the crystal class is different from that used in the integration, you should reprocess with the correct symmetry, ie with the correct cell constraints.

Systematic absences & space groups. These arise from translational symmetry operators, notably screw axes which lead to absences on axial reflections (in non-chiral crystals, glide planes lead to absences in 2-dimensional zones). They can thus be used to distinguish between different space groups within a chosen pointgroup. However, they are not always a reliable guide to the true space group, because there are relatively few axial reflections, and axes lying close to the spindle rotation axis may be only partly sampled or missing from the dataset altogether, so the information from the absences should be treated with caution. POINTLESS uses a Fourier analysis of I / σ values to estimate the probability of the translational element being present or not. For example, if the chosen point group might have a 2_1 screw dyad along the a axis, this would be indicated by presence of h00 reflections only when h is even(=2n). Then the one-dimensional Fourier transform of I / σ (or I) should peak at 1/2 in Fourier space, and the peak height at 1/2 relative to the origin is a measure of the strength of the screw. A probability of the presence of the screw is then calculated, using an error estimate derived from samples of the same number of observations of non-axial reflections, and again a Lorentzian distribution centred on the ideal value of 1. A similar analysis applies to 3-fold screws, and to glide planes, but 4-fold and 6-fold screws are more complicated, since there are multiple possible Fourier peaks, at 1/4 & 1/2 for a 4-fold, or 1/6, 1/3 & 1/2 for a 6-fold, and these are not independent. These can be treated by using a distribution based on a single deviation from all the 2 (4-fold) or 3 (6-fold) ideal peak values, considering the deviation as a "distance" in 2 or 3 dimensions.

In many cases, combining the probabilities from the rotational symmetry and from the systematic absences gives a unique choice of space group, but often several different space groups may need to be tried in the structure determination. POINTLESS tries to avoid over-confidence in its assignment of likelihood, which works in most cases, but it is occasionally fooled by close pseudo-symmetry.

Alternative indexing schemes

If the Laue symmetry is lower than the lattice symmetry, there are alternative indexing schemes which are different but equally valid, related by the rotational symmetry operators present in the lattice but not in the Laue group. These are the same conditions which allow merohedral twinning. For example, in Laue group P-3 (point group P3) there are four possible indexing schemes: (h,k,l);(-h,-k,l); (k,h,-l); (-k,-h,-l). As well as these exact cases, ambiguities may arise accidentally for special values of cell dimensions: for example, a monoclinic cell with β=90° will appear orthorhombic, leading to an alternative indexing as (-h,-k,l). Less obvious cases can occur with special relationships involving cell diagonals. For some examples, see http://www.ccp4.ac.uk/dist/html/reindexing.html

For the first crystal (or indexing), you are free to choose any of the alternatives, but subsequent indexing must match the original "reference" scheme. POINTLESS can check which scheme matches best in two ways: you can give a reference file HKLREF (which can now be either merged or unmerged), in which case the test data (HKLIN) will be

checked against the reference, and its space group will be assumed to be correct; or if you give multiple test data files (HKLIN) and no HKLREF file is defined, the first one will be treated as a reference for alternative indexing, but the combined data will still be tested for Laue group symmetry.

Examples

Two examples were given in Evans (2006) and they remain valid even though the scoring system has changed. Most crystals give a clear answer: uncertainties generally arise through pseudo-symmetry, including twinning, and the difficult case *(2)* illustrated here is not typical.

(1)Discriminating between orthorhombic groups with systematic absences

This case (Parker, unpublished) gave a clear indication of orthorhombic symmetry, Laue group Pmmm. Fourier analysis of the axial reflections gave a definitive suggestion that the space group was $P22_12_1$ (standard setting $P2_12_12$ with the reindexing operation (k,l,h))(table 1)

Axis	Number	Peak height at 1/2	SD	Probability	Reflection condition
2(1) [a]	39	-0.234	0.242	0.000	h00: h=2n
2(1) [b]	**27**	**0.997**	**0.176**	**0.970**	**0k0: k=2n**
2(1) [c]	**87**	**0.993**	**0.109**	**0.988**	**00l: l=2n**

Table 1 . *Systematic absence analysis*

(2)Pseudo-symmetry from incomplete pseudo-merohedral twinning

The true space group in this case (Sanchez Barrena, unpublished) is $P2_12_12_1$but all three cell lengths are about the same (79.2, 81.3, 81.2 Å) and the crystals have a variable amounts of twinning into the apparent point group 422(twinning operator k,h,-l). Table 2 shows the scores for the possible cubic symmetry operators for two crystals, a native crystal with about 20% twinning(refined twin fraction), and a more highly twinned SeMet crystal. Table 3 shows their scores for the different Laue groups: the native crystal gives the correct Pmmm group, but the program is fooled by twinning in the SeMet data into preferring Laue group P4/mmm.

	Native			SeMet		
Symmetry operator	**Likelihood**	**CC**	**R_{meas}**	**Likelihood**	**CC**	**R_{meas}**
Identity	0.953	0.97	0.066	0.949	0.97	0.076
2-fold (1 0 1)	0.058	0.22	0.466	0.055	0.04	0.605
2-fold (1 0 -1)	0.059	0.23	0.451	0.054	0.14	0.516
2-fold (0 1 -1)	0.063	-0.01	0.667	0.056	0.03	0.653
2-fold (0 1 1)	0.062	-0.01	0.671	0.054	0.05	0.636
2-fold (1 -1 0)	0.052	0.04	0.639	***0.713***	***0.83***	***0.155***
2-fold k (0 1 0)	**0.947**	**0.96**	**0.103**	**0.921**	**0.93**	**0.104**
2-fold (1 1 0)	0.051	0.05	0.631	***0.562***	***0.78***	***0.175***
2-fold h (1 0 0)	**0.944**	**0.95**	**0.138**	**0.916**	**0.92**	**0.111**
2-fold l (0 0 1)	**0.943**	**0.95**	**0.156**	**0.931**	**0.94**	**0.108**
3-fold (1 1 1)	0.059	0.00	0.766	0.058	0.02	0.639
3-fold (1 -1 -1)	0.058	0.01	0.776	0.057	0.03	0.800
3-fold (1 -1 1)	0.058	0.01	0.731	0.053	0.06	0.673
3-fold (1 1 -1)	0.058	0.01	0.798	0.059	0.02	0.838
4-fold h (1 0 0)	0.066	-0.02	0.731	0.064	-0.00	0.686
4-fold k (0 1 0)	0.056	0.21	0.463	0.054	0.05	0.586
4-fold l (0 0 1)	0.052	0.04	0.619	***0.539***	***0.77***	***0.175***

Table2. *Scores for potential symmetry elements for native and SeMet crystals. Both crystals show the dyads for orthorhombic symmetry (***bold***), but the SeMet crystal is more highly twinned (perhaps ~35%) than the native (~20%) also shows the tetragonal operators (****italic bold****)*

			Native					SeMet					
Rank	**Laue group**	**Reindex**	**Prob**	**Z-CC**	**CC+**	**CC-**	**R**	**Prob**	**Z-CC**	**CC+**	**CC-**	**R**	**Rank**
1	Pmmm	[h,k,l]	0.988	9.05	0.96	0.05	0.11	0.209	7.24	0.94	0.22	0.09	2
2	P 1 2/m 1	[k,l,h]	0.004	7.91	0.96	0.17	0.08	0.002	6.57	0.96	0.30	0.08	5
3	P 1 2/m 1	[l,h,k]	0.003	7.96	0.96	0.16	0.10	0.001	6.65	0.95	0.28	0.09	7
4	P 1 2/m 1	[h,k,l]	0.003	7.87	0.96	0.17	0.11	0.001	6.49	0.95	0.31	0.08	6
5	P 4/mmm	[l,k,-h]	0.000	4.58	0.53	0.07	0.32	0.000	2.61	0.54	0.27	0.29	13
6	P 4/m	[l,k,-h]	0.000	4.57	0.63	0.18	0.25	0.000	4.13	0.71	0.30	0.18	11
7	P 4/mmm	[k,h,-l]	0.000	6.53	0.66	0.01	0.24	**0.778**	**8.32**	**0.87**	**0.04**	**0.13**	**1**
8	P 4/m	[k,h,-l]	0.000	5.31	0.70	0.17	0.19	0.002	6.31	0.89	0.26	0.11	4
9	P -1	[h,k,l]	0.000	7.52	0.97	0.22	0.07	0.000	6.38	0.97	0.33	0.08	10
10	P 4/m	[h,l,-k]	0.000	4.35	0.62	0.18	0.25	0.000	3.01	0.63	0.33	0.23	12
11	P 4/mmm	[h,l,-k]	0.000	5.07	0.56	0.05	0.30	0.000	3.43	0.60	0.25	0.26	14

Table3. *Laue group rankings for the same crystals as in table 1. For the native crystal, the top rank solution is the correct Pmmm. For the SeMet crystal, P4/mmm is ranked higher because of the pseudo-symmetry from the twinning. Prob is the likelihood estimate, Z-CC is the net "Z-score" for CC, CC+ is for symmetry operators present in the Laue group, CC- for symmetry operators present in the cubic lattice but not in the Laue group, & R is the R-factor R_{meas}.*

Conclusions

In most cases, POINTLESS will give an unambiguous assignment of the Laue group, and often a good indication of the space group. Nevertheless, the results should always be treated with some caution, because of the possibility of pseudo-symmetry, which is not uncommon. In difficult cases, careful examination of the scores may lead to a decision different to that given by the program.

The options to combine multiple input files (from version 1.2.0) provides a more convenient method than the previous use of SORTMTZ, since it ensures that the files are on the same indexing system, and it adjusts the batch numbers if necessary to ensure that they are unique. POINTLESS is available from the CCP4 pre-release site or by anonymous ftp from ftp.mrc-lmb.cam.ac.uk/pub/pre/, and will be in future releases of CCP4. The program is under active development, the ultimate aim being to replace and extend all the scaling functions of Scala

Acknowledgements

I have been helped in the development of POINTLESS by many useful discussions with many people, including George Sheldrick, Ralf Grosse-Kunstleve, Airlie McCoy, Randy Read, Eleanor Dodson, Kevin Cowtan, Andrew Leslie, and Graeme Winter.

References

The Clipper C++ libraries for X-ray crystallography , Cowtan K. (2003) *IUCrComputing Commission Newsletter,***2**, 4-9

Scaling & assessment of data quality, Evans, P.R., (2006) *Acta Cryst.* D **62**, 82-82

*TheComputational Crystallography Toolbox: crystallographic algorithms in are usable software framework,*Grosse-Kunstleve, R.W., Sauter, N.K., Moriarty, N.W.& Adams, P.D (2002) *J.Appl.Cryst.* **35**, 126-136

XIA2 – a brief user guide

Graeme Winter, STFC Daresbury Laboratory Warrington WA4 4AD, United Kingdom
August 2007

Introduction

xia2 is an expert system for reducing diffraction data from macromolecular crystals, which makes use of existing data reduction software, including Mosflm (Leslie, 1992), Labelit (Sauter et Al, 2004), Pointless (Evans, 2006), CCP4 (Bailey, 1994) and XDS (Kabsch, 1993). There are two main programs of interest –

- xia2setup
- xia2

xia2setup is used to create the configuration file – which can of course be composed by hand, while xia2 performs the actual data reduction. In this article I will describe each of these programs, what they do and how they are used.

xia2setup

This program was written in response to feedback that writing the input file for xia2 was too time consuming. The setup program will read all of the headers of the images using diffdump – a program written using the Diffraction Image library (Remacle and Winter, 2006). This may take a moment or two for substantial data sets, so if you know what you are doing it may be simpler to just write the input file yourself. Come on – you know you want to!

The input file looks a little like this:

```
BEGIN PROJECT DEMO
BEGIN CRYSTAL INSULIN

BEGIN HA_INFO
ATOM S
NUMBER_PER_MONOMER 6
END HA_INFO

BEGIN WAVELENGTH SAD
WAVELENGTH 1.488
END WAVELENGTH SAD

BEGIN SWEEP SAD
WAVELENGTH SAD
DIRECTORY c:\data\insulin\images
IMAGE insulin_1_001.mar2300
END SWEEP SWEEP1

END CRYSTAL INSULIN
END PROJECT DEMO
```

This starts by establishing the project and crystal information – these will be used for harvesting and will end up in the resulting MTZ files. For each crystal we have some information about the wavelengths of data which were collected (the names of which will end up as WAVELENGTH_IDs in the harvest files and DATASETS in the MTZ files) and the sweeps which were measured for these wavelengths. If more than one sweep is assigned to a wavelength the reflections will be merged to form a single dataset – this is appropriate for low and high resolution passes, for instance. This example is for sulphur SAD data measured from cubic insulin on SRS station 7.2. This input file can be created with the command:

```
xia2setup –project DEMO –crystal INSULIN –atom s c:\data\insulin\images
```

which will write the results to the screen – you may want to pipe this to a file called e.g. demonstration.xinfo. If you have MAD data and you have left a scan file in this directory xia2setup will run Chooch and have a guess at the wavelength names, e.g. LREM, INFL, PEAK, HREM. If you do not specify a heavy atom and there is only one wavelength of data then xia2setup will guess that this is native data. If you have Labelit installed, this will be run to refine the beam centre prior to processing as well, so that you can ensure that the results look correct.

The way I would recommend using this is to run xia2setup to generate an input file, then open this file in your favourite editor to check that all of the names etc. are sensible. Note well – xia2setup will assume that all images in a directory come from the same crystal so if this is not the case you will need to edit the .xinfo file appropriately!

Now that we have the input file written we can move on to actually running the program.

xia2

xia2 is run very simply – you just need to run

```
xia2 –xinfo demonstration.xinfo
```

There are, however, a couple of useful command line options.

By default xia2 will run with the “2d” pipeline – that is, using Mosflm to perform the integration which uses two-dimensional profile fitting. This can be made explicit by adding “-2d” to the command line. Alternatively, if you wish to use XDS in the place of Mosflm use “-3d” on the command line. This tends to do a better job for highly mosaic data and lower resolution data (e.g. less than 2.5A resolution.)

The full breakdown of useful options is:

`[-quick]`	hurry – don’t make too much effort to be thorough
`[-migrate_data]`	move the data to e.g. /tmp on the local machine
`[-2d] or [-3d]`	select the integration program to use
`-xinfo foo.xinfo`	specify the input file

In addition, if you are using XDS you can specify a number of processors your machine has (xia2 will then use xds_par) – this is done with `"-parallel N"` where N is e.g. 2, 4. I do not recommend using –quick with -3d.

`-migrate_data` is useful if you are accessing the images over a relatively slow NFS connection – this will move them to a local disk which should result in a net improvement in processing time. Also useful if you are having problems with auto-mounted disks.

`-quick` cuts out some of the refinement of the resolution parameters but is useful if you want to get a data set while the sample is still available to confirm the pointgroup, quality of diffraction etc.

What does it do?

In a nutshell it will process all of your data for you, merging multiple sweeps within wavelengths together and scaling together data from multiple wavelengths e.g. for MAD experiments. The result is a reflection file suitable for immediate use in your favourite phasing pipeline, e.g. Mr BUMP, Happy, Phenix, Crank…

Along the way xia2 will figure out the correct pointgroup (with a *lot* of help from pointless) and have a good guess at the spacegroup. If it looks like the spacegroup is e.g. P 2 21 21 it will also reindex the data to the standard setting – P 21 21 2 in this case.

If you do not specify –quick – and you therefore allow xia2 to "do it's stuff" – each sweep will be integrated once to get an idea of where the observations are reasonable to, then again with this resolution limit to ensure that all of the profiles used are ok. Once all sweeps are integrated a preliminary scaling takes place, which may involve a number of "data shuffling" steps where the correct pointgroup and setting is assigned to each set of integrated intensities. From this a more reasonable resolution limit is determined – currently where I/sigma is about 2, which is then fed back to the integration if necessary to reprocess to a lower resolution. If necessary, the integration is repeated and then the full scaling takes place, where the scaling parameters are adjusted to optimise the error parameters and so on.

Once the integration and scaling steps are finished, the data are converted to F's using Truncate, merged together with CAD, have a single unit cell applied, twinning tests and so on are performed and a FreeR column is added. This is very helpful for MAD datasets as it can be quite time consuming and fiddly to do by hand.

If the results of the pointgroup analysis at the beginning of scaling indicate that the lattice used for integration was wrong, this will be eliminated from the set of possible indexing solutions and the processing repeated from the beginning. This is useful for cases where the lattice symmetry appears higher than the real crystal symmetry. Alternatively, for cases where you have e.g. a monoclinic cell with beta nearly but not exactly 90 degrees, the indexing solution may have already been eliminated based on the results of postrefinement.

Although it may take only a few minutes for a small data set (say 90 degree 1.8A native set) it has been known to take several hours for huge sets, for example with 2000 frames…

Output

While xia2 is working it will provide a running commentary of what is going on to the screen. If you specify “-debug” on the command line you will see an awful lot more of this! I don’t recommend this unless you are reporting a bug ;o)

The single most important part of the output is the list of citations you should include in your paper – xia2 is using lots of programs written by people who work really hard on them, and they should be acknowledged. This looks something like:

```
XIA2 used... ccp4 distl labelit mosflm pointless scala
Here are the appropriate citations (BIBTeX in xia-citations.bib.)
Bailey, S. (1994) Acta Crystallogr. D 50, 760--763
Evans, P.R. (1997) Proceedings of CCP4 Study Weekend
Evans, Philip (2006) Acta Crystallographica Section D 62, 72--82
Leslie, AGW (1992) Joint CCP4 and ESFEACMB Newsletter on Protein
Crystallography 26
Leslie, Andrew G. W. (2006) Acta Crystallographica Section D 62, 48--57
Sauter, Nicholas K. and Grosse-Kunstleve, Ralf W. and Adams, Paul D. (2004)
Journal of
Applied Crystallography 37, 399--409
Zhang, Z. and Sauter, N.K. and van den Bedem, H. and Snell, G. and Deacon, A.M.
(2006)
J. Appl. Cryst 39, 112--119
```

For those who use LaTeX, these citations are also provided in BibTex format in xia2-citations.bib. In addition to this xia2 includes a summary of the diffraction information from each wavelength – your standard “Table 1” stuff which Scala produces at the end of the log file:

```
For DEMO/INSULIN/SAD
High resolution limit                          1.78   5.64   1.78
Low resolution limit                           24.67  24.67  1.88
Completeness                                   97.8   98.9   84.9
Multiplicity                                   20.5   19.9   15.8
I/sigma                                        48.0   80.4   15.6
Rmerge                                         0.048  0.027  0.182
Rmeas(I)                                       0.052  0.032  0.195
Rmeas(I+/-)                                    0.051  0.028  0.194
Rpim(I)                                        0.011  0.007  0.047
Rpim(I+/-)                                     0.015  0.008  0.065
Wilson B factor                                19.49
Partial bias                                   -0.008 -0.008 -0.007
Anomalous completeness                         97.3   81.9
Anomalous multiplicity                         10.6   8.2
Anomalous correlation                          0.369  0.037
Anomalous slope                                1.473
Total observations                             153956 5223   14854
Total unique                                   7514   263    940
```

There is also a very concise summary of the integration for each run:

```
Integration status per image (60/record):

oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
"o"=> ok          "%" => iffy rmsd "!" => bad rmsd
"O"=> overloaded  "#" => many bad  "." => blank
```

This is nice for checking that the integration has gone well – for good data all you should see is "o" with the odd "%". If you have lots of "!" then there is probably something wrong.

For each step of processing the "final" log file is recorded in the LogFiles directory. The final reflection files can be found in the DataFiles directory, and all of the harvesting information is placed in the harvest directory.

Availability

xia2 is available from the ccp4 server at http://www.ccp4.ac.uk/xia - and is free to download and use, though you are responsible for supplying and correctly licensing CCP4 and optionally XDS and Labelit. To make installation more straightforward I include an "extras" package which includes the correct version of pointless and ipmosflm binaries – this is almost mandatory to install, unless you are in the habit of installing things from the CCP4 prerelease pages. These are distributed separately as xia2 is licensed as BSD, while the extras are licensed as per CCP4 license. New versions of xia2 are announced on the xia2bb – if you are interested in keeping up-to-date I would recommend you subscribe. The mailing list is also the appropriate forum for general xia2 chit-chat, although there is very little traffic.

Platforms

The following platforms are supported:

- 32 and 64 bit x86 linux
- Windows 2000/XP
- OS X PPC and Intel

All share the same packages – the setup scripts deal with the details.

Installation

Installation is relatively straightforward – unpack the tarballs and set XIA2_HOME correctly in the setup files. That's all! You will need to add

```
BASH                    . XIA2_HOME/setup.sh
(T)CSH                  source XIA2_HOME/setup.csh
Windows                 call XIA2_HOME/setup.bat
```

to your environment, but this is usually pretty easy. For Windows Francois Remacle has made a binary installer, which is probably the way to go ;o)

Acknowledgements and References

Development of xia2 is supported by e-HTPX, BioXHit and CCP4, and would not have happened without input from Harry Powell, Andrew Leslie, Nick Sauter, Phil Evans, Eleanor Dodson, Wolfgang Kabsch and many patient users. Also the following publications were critical in the development:

Bailey, S. (1994) Acta Crystallogr. D 50, 760--763

Evans, P.R. (1997) Proceedings of CCP4 Study Weekend

Evans, Philip (2006) Acta Crystallographica Section D 62,72--82

Kabsch, W. (1988) Journal of Applied Crystallography 21,67--72

Kabsch, W. (1988) Journal of Applied Crystallography 21,916--924

Kabsch, W. (1993) Journal of Applied Crystallography 26,795--800

Leslie, AGW (1992) Joint CCP4 and ESFEACMB Newsletter on Protein Crystallography 26

Leslie, Andrew G. W. (2006) Acta Crystallographica Section D 62, 48--57

Sauter, Nicholas K. and Grosse-Kunstleve, Ralf W. and Adams, Paul D. (2004) Journal of Applied Crystallography 37, 399--409

Zhang, Z. and Sauter, N.K. and van den Bedem, H. and Snell,G. and Deacon, A.M. (2006) J. Appl. Cryst 39, 112--119

smartie: a Python module for processing CCP4 logfiles

Peter Briggs

CCP4, CSE Department, STFC Daresbury Laboratory, Warrington WA4 4AD

Email: p.j.briggs@dl.ac.uk

---

1 Introduction: what is smartie?

smartie is a Python module containing classes and methods intended to provide tools for processing and interrogating CCP4 log files to more easily access generic information about the log file contents - in particular, information about which CCP4 programs have been run, warning messages that were generated, and any formatted tables of data that have been written.

smartie is built around the idea of providing a high-level view of a logfile that was loosely inspired by the HTML document object model (DOM) used for example in Javascript, where the DOM provides an interface for interrogating and manipulating the content of an HTML document. Smartie's equivalent "logfile object model" is much more modest, but still provides a way to drill down into some of the details of a logfile's content. In particular, smartie offers a nice way to easily interact with data in CCP4-formatted tables. The name "smartie" reflects one of the module's origins as the possible driver for a "smart logfile browser", although realising this aim is still some way off.

This article gives an overview of the ideas behind smartie, and by way of examples shows how to use its classes and functions:

- section 2 introduces smartie's "logfile object model" and explains how smartie attempts to impose structure on log files from programs and scripts.
- section 3 shows how smartie's classes and functions can be used to explore some of the "gross features" of a log file, including processing CCP4-formatted "tables" that are read by loggraph, and is generously peppered with working code examples
- section 4 briefly overviews two applications that use some of smartie's functions.

The final sections give overviews of smartie's limitations, possible future directions, and where to get hold of the code if you've been tempted to try it out after reading this article.

2 Background: an introduction to smartie's "logfile object model"

2.1 Anatomy of a logfile

155 scala.log is a log file from a run of the CCP4i "Scale and Merge Intensities" task. Principally this task runs the Scala program, but depending on the options set in the interface by the user a number of other programs may also be run before and after Scala. This log file will be used to help illustrate some of the features of smartie, and to introduce the "logfile object model".

As human beings who are more or less familiar with the output of CCP4i and CCP4 programs, we can scan through the log file by eye and recognise many different features, for example we are able to distinguish that this particular log file contains output from CCP4i (header and tail) plus logfile output from each of the programs that were run as part of the task script. We can see that some of the programs also generated some warning messages and tables (which we could view graphically using the loggraph program), and that the program output also contains CCP4 summary tags and HTML markup. Although the file is essentially one big "blob" of text, we are nonetheless able to impose a conceptual structure on the file and thus find our way around.

Similarly, smartie tries to impose some structure onto the log file by attempting to interpret it as a sequence of **fragments** - each fragment being a distinct section of the log file that is delimited by some generic feature that smartie recognises. Examples of these generic features include program banners and termination messages (such as for example those shown in figure 1), and particular forms of messaging from CCP4i.

(a)
```
##################################################################
##################################################################
##################################################################
### CCP4 6.0: SORTMTZ            version 6.0       : 06/09/05##
##################################################################
User: pjx  Run date: 13/ 7/2007 Run time: 17:09:59

Please reference: Collaborative Computational Project, Number 4. 1994.
"The CCP4 Suite: Programs for Protein Crystallography". Acta Cryst. D50, 760-763.
as well as any specific reference in the program write-up.
```

(b)
```
 SORTMTZ:  Normal termination
Times: User:       0.1s System:    0.0s Elapsed:     0:00
```

Figure 1: example of a CCP4 program banner (a) and termination message (b).

After identifying a fragment, smartie tries to make its identification more specific. If a fragment has features that match the output of a CCP4 program then classifies this as a **program**; if it looks like output from CCP4i then this is classified as **CCP4i information**. It is also possible that a fragment cannot be classified further - in this case smartie just keeps this as a fragment.

For a program fragment, smartie examines the features that it recognises and tries to extract some additional (generic) information - for example the program name and version, the user who ran it, the date and time and so on. It also extracts information about any keywords or file openings that are reported in standard formats, example of which are shown in figure 2.

(a)

```
Data line--- title Example run with aucn data
Data line--- name project DMSO crystal DMSO dataset red_aucn
Data line--- exclude EMAX     10.0
Data line--- partials     check     test 0.95 1.05     nogap
Data line--- intensities PROFILE     PARTIALS
Data line--- final PARTIALS
Data line--- scales     rotation SPACING 5     secondary 6     bfactor ON     BROTATION SPACING 20
Data line--- UNFIX V
Data line--- FIX A0
Data line--- UNFIX A1
Data line--- initial MEAN
Data line--- tie surface 0.001
Data line--- tie bfactor 0.3
Data line--- cycles 10 converge 0.3 reject 2
Data line--- output AVERAGE
Data line--- print brief nooverlap
Data line--- RSIZE 80
```

(b)

```
OPENED INPUT MTZ FILE
Logical Name: HKLIN   Filename: /home/pjx/CCP4_REMOTE/ccp4/examples/toxd/toxd.mtz
```

Figure 2: example of keyword echoing (a) and file opening report (b) from a CCP4 program logfile.

Log files can also contain formatted **tables** that are marked up in the CCP4 "$TABLE" format, and which can be interpreted by programs such as loggraph in order to produce graphs. Similarly the log file may also contain warning and informational messages **keytext messages** marked up in the CCP4 "$TEXT" format. (Keytext messages are typically produced by calls within CCP4 programs to either the Fortran library subroutine CCPERR, or the C library function ccperror - see the *CCP4LIB* documentation at http://www.ccp4.ac.uk/dist/html/ccplib.html or the *ccp4_general.c File Reference* at http://www.ccp4.ac.uk/dist/html/C_library/ccp4__general_8c.html for more information.)

Both these types of feature are described in the CCP4 program documentation, for example at http://www.ccp4.ac.uk/dist/html/loggraphformat.html, and examples of each are also shown in figure 3.

```
$TABLE: Rfactor analysis, stats vs cycle  :
$GRAPHS:<Rfactor> vs cycle :N:1,2,3:
:FOM vs cycle :N:1,4:
:-LLG vs cycle :N:1,5:
:Geometry vs cycle:N:1,6,7,8:
$$
      Ncyc   Rfact   Rfree     FOM         LLG  rmsBOND  rmsANGLE rmsCHIRAL $$
$$
         0   0.206   0.178   0.889     16260.1    0.030    3.367    0.248
         1   0.189   0.196   0.868     15382.6    0.022    2.213    0.136
         2   0.181   0.204   0.856     15228.2    0.022    2.060    0.122
         3   0.177   0.210   0.849     15206.6    0.022    2.048    0.121
         4   0.175   0.215   0.843     15211.9    0.022    2.044    0.122
         5   0.174   0.221   0.839     15215.6    0.022    2.044    0.124
         6   0.174   0.222   0.837     15214.7    0.022    2.039    0.126
         7   0.174   0.225   0.835     15220.9    0.022    2.036    0.127
         8   0.173   0.227   0.833     15230.7    0.022    2.030    0.128
         9   0.173   0.228   0.832     15234.0    0.022    2.028    0.130
        10   0.172   0.229   0.831     15233.9    0.022    2.030    0.131
$$
```

(a)

```
$TEXT:Warning: $$ comment $$
WARNING:  **** Beware! - Cell dimensions could permit Twinning ****
$$
```

(b)

Figure 3: examples of tabular data formatted into the CCP4 "$TABLE" markup for loggraph (a), and an informational message marked up in the "$TEXT" format (b).

Typically both tables and keytext messages are generated by programs, however in principle they can fall anywhere in the log file and so do not themselves represent specific fragments of the file. Instead they are considered to be a part of the fragment that encloses them in the file.

Finally: logfiles from CCP4 programs can also contain "summary tags" and HTML formatting. An example logfile chunk containing examples of both is shown in figure 4. Smartie keeps track of the start and end summary tags in the log file as a whole but doesn't (yet) assign them to specific fragments. (HTML formatting is unfortunately more of a nuisance than anything to smartie and so is generally ignored.)

```
<B><FONT COLOR="#FF0000"><!--SUMMARY_BEGIN-->

    WILSON PLOT for Ranges   35 -  60
    Resolution range:  3.9460  3.0127
<pre>
  LSQ Line Gradient =         -48.178608
 Uncertainty in Gradient  =      0.2197E+01
  X Intercept             =     -0.4395E+01
 Uncertainty in Intercept =      0.9699E-01

 For a wilson plot          B          =  -  gradient
                          SCALE        = exp( - intercept).

 Least squares straight line gives:   B  = 48.179        SCALE  =  81.04675
    where  F(absolute)**2 = SCALE*F(observed)**2*EXP(-B*2*SINTH**2/L**2)

<!--SUMMARY_END--></FONT></B>
```

Figure 4: example of a section of logfile enclosed in "summary tags", specifically <!--SUMMARY_BEGIN--> and <!--SUMMARY_END-->, and also containing additional HTML markup.

2.2 Introducing smartie's classes for the logfile object model

Once smartie has processed a log file, it is represented by a **logfile object** (this object and the others introduced here are described in more detail in section 3). The individual sections of the original file (fragments, programs and CCP4i info blocks) are also each represented by distinct objects, with the objects describing programs and CCP4i info blocks being derived from the generic fragment object.

The logfile object holds a master list of all the fragments that smartie recognised, in the order that they were encountered in the file. It also keeps a smaller list holding the subset of fragments that are programs, and another list of the subset that were CCP4i info blocks. Tables, keytexts and summaries are represented by specialised objects that are not derived from any of the other classes of objects. The logfile object keeps a master lists of all the tables, all the keytexts and all the summaries found in the file. In addition each fragment also has its own lists of the tables and keytexts that were found in that fragment.

To get a summary printout of smartie's analysis of the 155_scala.log file, you can run the command:

```
> python smartie.py 155_scala.log
```

which will generate the summary output shown in figure 5.

The summary gives an overview of what smartie found in the log file. First it lists the "fragments" that it found - that is, all the discrete blocks of output. In this case, all the fragments are also program logs, so the list of "programs" that follows is the same. Examination of the `scala.exam` script shows that it does indeed run the four programs listed.

For the list of programs, the summary of each individual program log includes some of the information that has been extracted from the program banner and termination messages. It also lists the names of each of the loggraph-formatted tables within each program log file, and information on the keytext messages found (for example, a warning from TRUNCATE about the possibility of twinning).

After the program list there are also lists of all the keytext messages and tables found in the log file as a whole, plus any CCP4i messages that were found (in this case none). If the logfile had been generated from a run of a CCP4i task then it's likely that there would have been additional messages reported from CCP4i.

This example is intended to give a basic idea of how smartie represents a log file. The following sections cover using the Python classes and functions to examine log file content in more detail.

```
Running test on logparser code
command line: ['/home/pjx/PROGRAMS/smartie/smartie.py', '155_scala.log']
**** Parsing file "155_scala.log"
Summary for 155_scala.log

This is a CCP4i logfile

4 logfile fragments

Fragments:
        CCP4i info
        Program: SORTMTZ
        Program: Scala
        Program: TRUNCATE

3 program logfiles

Programs:
        SORTMTZ v6.0    (CCP4 6.0)
        Terminated with: Normal termination

        Scala   v6.0    (CCP4 6.0)
        Terminated with: ** Normal termination **

                Tables:
                Table: ">>> Scales v rotation range, red_aucn"
                Table: "Analysis against Batch, red_aucn"
                Table: "Analysis against resolution , red_aucn"
                Table: "Analysis against intensity, red_aucn"
                Table: "Completeness, multiplicity, Rmeas v. resolution,
red_aucn"
                Table: "Correlations within dataset, red_aucn"
                Table: "Run     1, standard deviation v. Intensity, red_aucn"

        TRUNCATE        v6.0    (CCP4 6.0)
        Terminated with: Normal termination

                Keytext messages:
                Warning: "WARNING:  **** Beware! - Cell dimensions could permit
Twinning ****"
                Warning: "WARNING:  ***Beware-serious ANISOTROPY; data analyses
may be invalid ***"

                Tables:
                Table: "Wilson Plot - Suggested Bfactor 52.3"
                Table: "Acentric Moments of E for k = 1,3,4,6,8"
                Table: "Centric Moments of E for k = 1,3,4,6,8"
                Table: "Cumulative intensity distribution"
                Table: "Amplitude analysis against resolution"
                Table: "Anisotropy analysis (FALLOFF).  Example run with aucn
data"


CCP4i messages in file:
        CCP4i info: "Sorting input MTZ file
/home/pjx/PROJECTS/myProject/aucn.mtz"

Tables in file:
        Table: ">>> Scales v rotation range, red_aucn" (6 rows)
        Table: "Analysis against Batch, red_aucn" (6 rows)
        Table: "Analysis against resolution , red_aucn" (10 rows)
        Table: "Analysis against intensity, red_aucn" (13 rows)
```

```
        Table: "Completeness, multiplicity, Rmeas v. resolution, red_aucn" (10
rows)
        Table: "Correlations within dataset, red_aucn" (10 rows)
        Table: "Run     1, standard deviation v. Intensity, red_aucn" (13 rows)
        Table: "Wilson Plot - Suggested Bfactor 52.3" (60 rows)
        Table: "Acentric Moments of E for k = 1,3,4,6,8" (60 rows)
        Table: "Centric Moments of E for k = 1,3,4,6,8" (60 rows)
        Table: "Cumulative intensity distribution" (11 rows)
        Table: "Amplitude analysis against resolution" (60 rows)
        Table: "Anisotropy analysis (FALLOFF).  Example run with aucn data" (60
rows)

Keytext messages in file:
        Warning: "WARNING:  **** Beware! - Cell dimensions could permit Twinning
****"
        Warning: "WARNING:  ***Beware-serious ANISOTROPY; data analyses may be
invalid ***"


Time: 1.37
```

Figure 5: summary output from smartie after processing the log file from the standard CCP4 Scala example script

3 Using smartie to look at CCP4 log files

The following sections introduce smartie's features by example, with a set of code fragments and resulting output. If you're familiar with the Python programming language then it should be simple to start using smartie and the examples should be straightforward to follow. If you're not familiar with Python then there are lots of resources at www.python.org to help you get started.

3.1 Getting started with smartie

The logfile object lies at the heart of smartie. To create a new logfile object from a from a log file (say the same `155_scala.log` that was used in the previous section), we would use the following Python code:

```
>>> import smartie
>>> logfile = smartie.parselog("155_scala.log")
```

which will create and populate a new logfile object. Once we have this object, we can use its methods to probe the content of the file - for example, to generate the same summary as shown in figure 5 we can use smartie's "summarise" function:

```
>>> smartie.summarise(logfile)
```

The following sections give some more interesting examples that illustrate the kind of information that smartie can tell you about a log file.

3.2 Examining the structure of the log file

Once a smartie logfile object has been created it is very easy to get data about its structure and content. To begin with, you might be interested in the fragments that smartie found. To look up the total number of fragments:

```
>>> logfile.nfragments()
4
```

A fragment is any particular chunk of log file that smartie recognised, and is represented by a fragment or fragment-based object. To acquire the object representing a particular fragment, use the logfile object's "fragment" method - this example returns the fragment object for the first fragment:

```
>>> fragment = logfile.fragment(0)
>>> print fragment
<smartie.ccp4i_info instance at 0x2abed92749e0>
```

Only certain types of fragment may be of interest for a particular application. The "isccp4i_info" and "isprogram" methods of the fragment-based objects can be used to find out about the object types:

```
>>>
>>> logfile.fragment(0).isccp4i_info()
True
>>> logfile.fragment(0).isprogram()
False
>>> logfile.fragment(1).isccp4i_info()
False
>>> logfile.fragment(1).isprogram()
True
```

Examination of the 155_scala.log file shows that this is as we might expect - the log file starts with a preamble message from the CCP4i script before running the first program. If we were curious about what message CCP4i actually wrote then we could look this up:

```
>>> logfile.fragment(0).message
'Sorting input MTZ file /home/pjx/PROJECTS/myProject/aucn.mtz'
```

However it's more likely that you're interested in the programs that ran. While you could figure out which fragments are program logs by examining each fragment as shown above, smartie shortcuts this by providing a methods in the logfile object specifically for examining the programs. These are examined in more detail in the next section.

3.3 Getting information about the program log output

The logfile object introduced in the previous section provides the "nprograms" method that returns the number of program fragments found in the log file, for example for 155_scala.log:

```
>>> logfile.nprograms()
3
```

This is what we should expect - there is log file output from three programs in the 155_scala.log example file.

We can get some information about the individual program fragments. We use the "program" method returns the object representing a specific program fragment, and then we can access its data via its attributes and methods. For example:

```
>>> program = logfile.program(0)
>>> program.name
'SORTMTZ'
>>> program.version
'6.0'
>>> program.termination_message
'Normal termination'
```

The attributes are populated using data that has been extracted from the program banners and termination messages in the logfile. For CCP4 programs there are a number of available data items. To get a list of all the available attributes, use the program's "attributes" method - for example:

```
>>> program.attributes()
['termination_name', 'systemtime', 'startline', 'rundate', 'name',
'usertime', '
banner_text', 'elapsedtime', 'termination_text', 'source_file', 'user',
'version
', 'date', 'ccp4version', 'endline', 'runtime', 'nlines',
'termination_message']
```

Each of these attributes is described in more detail in the module documentation for smartie. (Note that the "attributes" method is available for any fragment-like object, although the attributes themselves can vary even between different instances of the same class.)

As described in the previous section, program log file fragments can also contain formatted tables and "keytext" warning messages (in fact this is true of any kind of fragment).

```
>>> logfile.program(1).name
'Scala'
>>> logfile.program(1).ntables()
7
>>> logfile.program(2).name
'TRUNCATE'
>>> logfile.program(2).nkeytexts()
2
```

Basically this is telling us that there were 7 tables in the logfile for the second program run (Scala) and 2 keytext warnings for the third (Truncate).

Working with tables is covered in more detail in the next section, as this is currently probably the single most useful aspect of smartie's functionality. The keytexts are much simpler; the program object has a "keytext" method that returns a keytext object, and the keytext's attributes can be retrieved. For example, the first warning in the Truncate log output in the file looks like:

```
$TEXT:Warning: $$ comment $$
WARNING:  **** Beware! - Cell dimensions could permit Twinning ****
$$
```

To access this data in smartie:

```
>>> keytext = logfile.program(2).keytext(0)
>>> keytext.message()
'WARNING:  **** Beware! - Cell dimensions could permit Twinning ****'
```

The program objects also have lists of any keywords and file opening operations that were reported in the logfile, and this data can be accessed using the "keywords", "logicalnames" and "logicalnamefile" methods.

To get a list of the reported keywords for the first program in the 155_scala.log example:

```
>>> logfile.program(0).name
'SORTMTZ'
>>> logfile.program(0).keywords()
['ASCEND', 'H K L M/ISYM BATCH']
```

Each keyworded input line is returned as an item in the list; smartie's "tokeniser" function could be used to further process each line in order to break it up into discrete tokens (see section 3.6.2).

File opening reports in CCP4 log files look like this example:

```
Logical Name: HKLOUT   Filename: /tmp/pjx/PROJECT_155_4_mtz_red_aucn.tmp
```

A "logical name" (in this case "HKLOUT") is associated with a physical filename when the program is run (more information about logical names and filenames can be found in the CCP4 manual, chapter 3 section 3.2.1 "Command line arguments/file connexion").

Here is an example of getting a list of the logical names found in a particular program log, using the "logicalnames" method:

```
>>> logfile.program(1).name
'Scala'
>>> logfile.program(1).logicalnames()
['HKLIN', 'HKLOUT', 'SYMINFO']
```

The file name associated with any logical name can then be retrieved using the "logicalnamefile" method:

```
>>> logfile.program(1).logicalnamefile("HKLIN")
'/home/pjx/PROJECTS/myProject/aucn_sorted.mtz'
```

Finally, the full text of a program fragment (or any other fragment) can be retrieved from the source file using the "retrieve" method. For example, the following example would retrieve the text of the Sortmtz log output from 155_scala.log:

```
>>> print logfile.program(0).retrieve()
 ###############################################################
 ###############################################################
 ###############################################################
 ### CCP4 6.0: SORTMTZ             version 6.0       : 06/09/05##
 ###############################################################
 User: pjx  Run date: 13/ 7/2007 Run time: 17:09:59

 ...
```

Smartie's "strip_logfile_html" function is useful if you wish to remove any HTML tags in the text whilst preserving features such as CCP4 formatted tables (see section 3.6.1).

3.4 Working with tables in log files

Smartie's table objects represent CCP4 formatted $TABLES and provide a variety of methods that can be used to extract data from these tables and reformat them for output.

To look at working with the tables we'll introduce a new example log file, 156_refmac5.log. This example contains the log file from a CCP4i task run of the refinement program Refmac5. To start examining this log file using smartie we must make a new logfile object:

```
>>> import smartie
>>> logfile = smartie.parselog("156_refmac5.log")
```

Often after running Refmac5 you are interested in seeing the contents of the final table that the program writes out (the "Rfactor analysis, stats vs cycle" table) since this gives details of how the refinement behaved over each cycle and can give an idea of the quality of the result after running the program. For 156_refmac5.log the table looks like this:

```
$TABLE: Rfactor analysis, stats vs cycle  :
$GRAPHS:<Rfactor> vs cycle :N:1,2,3:
:FOM vs cycle :N:1,4:
:-LLG vs cycle :N:1,5:
:Geometry vs cycle:N:1,6,7,8:
$$
      Ncyc   Rfact   Rfree     FOM          LLG  rmsBOND  rmsANGLE rmsCHIRAL $$
$$
         0   0.228   0.221    0.851       88700.2    0.028    4.533    0.163
         1   0.184   0.201    0.872       85078.6    0.029    2.509    0.178
         2   0.169   0.195    0.878       83708.6    0.030    2.448    0.194
         3   0.162   0.193    0.879       83089.2    0.031    2.431    0.200
         4   0.158   0.192    0.879       82787.4    0.031    2.404    0.202
         5   0.155   0.190    0.879       82619.6    0.031    2.378    0.202
 $$
```

To illustrate working with smartie's table object we will show how smartie can be used to easily get information from this table - we will find out the number of cycles of refinement and the initial and final values of the R factor and of R_{free}.

First, we need to locate the table object in the logfile, since there are many tables in the file (and all of these are in the Refmac log itself):

```
>>> logfile.ntables()
7
>>> logfile.nprograms()
1
>>> logfile.program(0).name
'Refmac_5.3.0040'
>>> logfile.program(0).ntables()
7
```

(Aside: note that the same methods for tables - for example "ntables" - that are available in logfile objects are also available in fragment and program objects. So although the remaining examples only show the methods of the program objects, they can be used from the other objects in exactly the same way.)

Since we know the title of the table we're looking for, we can use the program object's "tables" method to try and look it up. The "tables" method takes a title (or a fragment of a title) and returns a list of table objects that match. For example, the table we're interested in is called "Rfactor analysis, stats vs cycle":

```
>>> tables = logfile.program(0).tables("Rfactor analysis")
>>> len(tables)
1
>>> table = tables[0]
>>> table.title()
'Rfactor analysis, stats vs cycle'
```

The search for the partial title returns a list of all matching table objects - in this case there is only one (but in the general case there could be more than one, or none). The "title" method of the table object returns the full title of the table.

Having acquired the table object, we can then get some of the data values from the columns. For example, if we wanted a list of the values in the "Ncyc", "Rfact" or "Rfree" columns, then we would use the "col" method of the table object to return the list of values in each:

```
>>> table.col("Ncyc")
[0, 1, 2, 3, 4, 5]
>>> table.col("Rfact")
[0.22800000000000001, 0.184, 0.16900000000000001, 0.16200000000000001,
0.158, 0.155]
>>> table.col("Rfree")
[0.221, 0.20100000000000001, 0.19500000000000001, 0.193, 0.192, 0.19]
```

(Note that the string values in the original table have been converted by smartie to floating point values.)

Since the "col" method returns a normal Python list, the results can be interrogated using normal methods of accessing list items - for example, to get the last value in the "Ncyc" column (which corresponds to the total number of cycles of refinement) we can use the standard Python idiom for acquiring the last element in a list:

```
>>> table.col("Ncyc")[-1]
5
```

Something else that we might wish to do is to calculate the change in the R factor and R_{free} values over the course of the refinement:

```
>>> delta_Rfact = table.col("Rfact")[-1] - table.col("Rfact")[0]
>>> print str(delta_Rfact)
-0.073
>>> delta_Rfree = table.col("Rfree")[-1] - table.col("Rfree")[0]
>>> print str(delta_Rfree)
-0.031
```

It would be straightforward to write a small Python program that combined these code fragments to return the change in R factors given any log file from Refmac5.

The table object also offers a set of methods for outputting the table in different formats: "show" prints a basic text representation without any of the $TABLE markup for loggraph, "loggraph" generates the table including the $TABLE markup, and "html" generates an HTML table that is suitable for inclusion in a webpage. The results of these three methods are shown in figure 6 below.

(a)

Ncyc	Rfact	Rfree	FOM	LLG	rmsBOND	rmsANGLE	rmsCHIRAL
0	0.228	0.221	0.851	88700.2	0.028	4.533	0.163
1	0.184	0.201	0.872	85078.6	0.029	2.509	0.178
2	0.169	0.195	0.878	83708.6	0.03	2.448	0.194
3	0.162	0.193	0.879	83089.2	0.031	2.431	0.2
4	0.158	0.192	0.879	82787.4	0.031	2.404	0.202
5	0.155	0.19	0.879	82619.6	0.031	2.378	0.202

(b)

```
$TABLE: Rfactor analysis, stats vs cycle:
$GRAPHS
 :<Rfactor> vs cycle :N:1,2,3:
 :FOM vs cycle :N:1,4:
 :-LLG vs cycle :N:1,5:
 :Geometry vs cycle:N:1,6,7,8:
$$
  Ncyc   Rfact   Rfree    FOM      LLG   rmsBOND   rmsANGLE   rmsCHIRAL $$
$$
     0   0.228   0.221   0.851  88700.2     0.028      4.533       0.163
     1   0.184   0.201   0.872  85078.6     0.029      2.509       0.178
     2   0.169   0.195   0.878  83708.6      0.03      2.448       0.194
     3   0.162   0.193   0.879  83089.2     0.031      2.431         0.2
     4   0.158   0.192   0.879  82787.4     0.031      2.404       0.202
     5   0.155    0.19   0.879  82619.6     0.031      2.378       0.202
$$
```

(c)

Ncyc	Rfact	Rfree	FOM	LLG	rmsBOND	rmsANGLE	rmsCHIRAL
0	0.228	0.221	0.851	88700.2	0.028	4.533	0.163
1	0.184	0.201	0.872	85078.6	0.029	2.509	0.178
2	0.169	0.195	0.878	83708.6	0.03	2.448	0.194
3	0.162	0.193	0.879	83089.2	0.031	2.431	0.2
4	0.158	0.192	0.879	82787.4	0.031	2.404	0.202
5	0.155	0.19	0.879	82619.6	0.031	2.378	0.202

Figure 6: examples of outputing the table data using different table object output methods
(a) Using table.show() gives a basic text printout.
(b) Using table.loggraph() regenerates the table with CCP4 $TABLE loggraph markup
(c) using table.html() generates a HTML marked-up table that can be incorporated into a webpage and viewed in a web browser.

There is also a "jloggraph" method which generates the same output as the "loggraph" method, but with additional HTML tags that allow the table to be viewed in the JLogGraph Java applet when loaded into a web page (with the caveat that the currently distributed JLogGraph code doesn't always work with smartie-generated tables - this will be fixed for future releases of JLogGraph).

3.5 Working with summary tags in logfiles

As mentioned in the overview of log files earlier in this article, smartie also keeps track of the start and end summary tags (<--SUMMARY_BEGIN--> and <--SUMMARY_END--> respectively) found in the log file. This information is stored in a list of "summary" objects stored in the log file.

To return to the 155_scala.log example file used earlier on, the number of summaries can be obtained using the "nsummaries" method of the logfile object:

```
>>> logfile.nsummaries()
32
```

Individual summary objects can be fetched from the logfile object using the "summary" method, although these objects aren't particularly interesting in themselves: the most useful thing you can do with them is invoke their "retrieve" method to fetch the actual text enclosed in the summary tags.

As an example, to fetch the 3rd summary object from 155_scala.log:

```
>>> logfile.summary(2).retrieve()
'<'B><FONT COLOR="#FF0000"><!--SUMMARY_BEGIN-->\n<li><a
href="#commandSORTMTZ">Co
mmand Input</a>\n<<li><a href="#inputSORTMTZ">Input File
Details</a>\n<li><a href
="#outputSORTMTZ">Output File Details</a>\n<!--SUMMARY_END--
></FONT></B>\n'
```

This isn't particularly pretty, however smartie's "strip_logfile_html" function (see section 3.6) can be used to clean it up a bit:

```
>>> smartie.strip_logfile_html(logfile.summary(2).retrieve())
'Command Input\nInput File Details\nOutput File Details\n\n'
```

One obvious application of this would be to write out all the summaries from a log file in one go, for example:

```
>>> for i in range(0,logfile.nsummaries()):
...     print smartie.strip_logfile_html(logfile.summary(i).retrieve())
...
```

3.6 Some other useful functions in Smartie

Smartie has a number of supporting functions, most of which are not really very interesting outside of the module. However two of the functions may be of use to other applications: "strip_logfile_html" and "tokenise", and so are described in more detail in the following sections.

3.6.1 smartie.strip_logfile_html(): cleaning up logfiles

HTML mark up was added to a number of CCP4 programs several years ago, which allowed log files from those programs to be viewed in a web browser. Within smartie however the HTML can sometimes be something of a nuisance, and so the strip_logfile_html function can be used to remove it from arbitrary text. As an example, here is a fragment of text from 155_scala.log containing HTML markup being run through the function:

```
>>> print text
<a name="commandsScala"><h3> Input keyworded commands (click for
documentation):</h3></a>
<a href="/home/pjx/CCP4I_uWORKSHOP/ccp4/html/scala.html#title">TITLE</a>
 Example run with aucn data
<a
href="/home/pjx/CCP4I_uWORKSHOP/ccp4/html/scala.html#name">NAME</a>

>>> print smartie.strip_logfile_html(text)
 Input keyworded commands (click for documentation):
TITLE
 Example run with aucn data
NAME
```

Where strip_logfile_html is most useful however is in dealing with $TABLES that have been marked up with the Jloggraph applet code. In these cases the entire table is printed inside an attribute of a "param" tag, and this function is careful to extract the text table in this case.

3.6.2 smartie.tokenise(): CCP4-style line tokeniser

The tokenise function in smartie tries to replicate the basic functionality of the keyword parser in the CCP4 libraries. The naive solution to tokenising these lines (i.e. breaking them up into discrete tokens) is to split on white space; however this approach fails for quoted tokens which themselves contain whitespace, as shown in the example below:

```
>>> print line
HKLIN "C:\Documents and Settings\pjb93\My Documents\rnase.mtz"
>>> line.split()
['HKLIN', '"C:\\Documents', 'and', 'Settings\\pjb93\\My',
'Documents\\rnase.mtz"']
>>> print smartie.tokenise(line)
```

```
['HKLIN', '"C:\\Documents and Settings\\pjb93\\My Documents\\rnase.mtz"']
```

Smartie's tokeniser function deals with this in a fashion that is more consistent with the core CCP4 suite, and may be useful in some cases.

4 Applications currently using smartie

4.1 starKey

starKey is a small program that has been developed in order to produce a summary of the programs used by jobs run within CCP4i, and was the original motivator for developing smartie. Although the CCP4i job database stores information on each task that was run in a project, it doesn't hold information on the specific underlying programs that were used in each run. So starKey uses smartie to discover this information *a posteri* from the logfiles alone.

starKey has been developed as part of CCP4's contribution to the BIOXHIT project; more information can be found at http://www.ccp4.ac.uk/projects/bioxhit.html, and starKey (amongst other software) can be downloaded from http://www.ccp4.ac.uk/projects/bioxhit_public/

4.2 MrBUMP

MrBUMP is an automated system for performing molecular replacement. Part of its operation involves running CCP4 programs such as Refmac5 and analysing the output to determine whether a putative molecular replacement solution is worth pursuing. Smartie is used within MrBUMP to extract data from the formatted tables, and in some instances to reformat the tables for output in a master log file.

MrBUMP can be obtained from the website at http://www.ccp4.ac.uk/MrBUMP/

5 Limitations and Possible Future Development

The development of smartie so far has been somewhat haphazard and there are many places where functionality is either missing or is not particularly well implemented. In particular the classes and functions for handling tables could be greatly improved (for example, in the course of writing this article it became apparent that there is no way to list the names of columns in a table).

However possibly the most serious limitation at present is that smartie cannot recognise a wider range of "non-generic" features from specific programs - for example, information that is output only from Scala or Refmac5. Capturing this specific kind of information would be a lot of work and it is not clear that it is a practical goal; however some experiments in doing this are taking place. If the data could be captured then smartie could realistically be used as the engine for a new generation of CCP4 log file browser.

Another limitation is that smartie doesn't currently recognise output from programs that don't have standard CCP4 start or end banners (an exception to this is that it does already recognise the banners from SHELXC, D and E). Some experimental code is already extant that investigates overcoming these two limitations but is not generally available, however if you are interested then it can be provided on request.

Aside from these, there are a number of possible directions that future development of smartie could take, for example the "real-time" processing of log files, or the provision of additional methods for rendering the table objects graphically for example via Tk. However since much of the development so far has been driven by functionality requests from end users, this development model is likely to continue as the preferred way forward. Therefore I would welcome any requests from potential or actual users of smartie about how the code should be developed.

6 Conclusion

smartie is a Python module containing classes and methods for performing simple processing of CCP4 log files in order to extract basic information. Smartie provides functions to break down the structure of a log file to identify the programs that were run and to access information that has been marked up in special formats such as CCP4's $TABLE tags.

In this article I have given a practical overview of how smartie's classes and functions can be used to obtain information on the "gross features" within a CCP4 log file, along with examples of working code that will hopefully allow people to see how smartie might be used in their own applications.

7 Availability

The current version of smartie at the time of writing is 0.0.9. The latest version of smartie can always be downloaded via FTP from ftp://ftp.ccp4.ac.uk/pjx/ccp4/smartie/

The distribution currently includes the smartie.py module plus a number of example logfiles that can be used to try out the code on.

Finally, if you are interested in using smartie, or have suggestions about it could be developed or used in future, then I would be very interested in working with you - please contact me at **p.j.briggs@dl.ac.uk**.

8 Acknowledgements

PJB would like to acknowledge funding from STFC Daresbury Laboratory via the CCP4 project and from the BIOXHIT project, which is funded by the European Commission within its FP6 Programme, under the thematic area "Life sciences, genomics and biotechnology for health", contract number LHSG-CT-2003-503420.

PJB would also like to acknowledge the valuable testing performed by Wanjuan Yang, and the very valuable suggestions from Ronan Keegan which were very helpful in developing smartie and in particular the table handling classes.

Finally, the figures in this article were prepared using the Gimp and the "convert" program from the Imagemagick suite.

Project Tracking System for Automated Structure Solution Software Pipelines

Wanjuan Yang, Ronan Keegan and Peter Briggs

CCP4, CSE Department, STFC Daresbury Laboratory, Warrington WA4 4AD

1. Introduction

In recent years, there has been an increasing number of projects focusing on the development of automated structure determination software "pipelines" covering various aspects of the post-data collection stages of structure solution by protein X-ray crystallography (PX). Examples of these include MrBUMP[1], Autorickshaw[2], Crank[3], HAPPy[4], XIA2[5].

Generally pipelines work in a similar way to how an experienced scientist would work – they make choices about which programs to run and try to select optimal parameters by analysis of the input data and the outputs of the underlying software. The automated systems are therefore of help to both less experienced users, who may get better results than by using the same programs manually, and for expert users where the automated procedure can perform routine tasks quickly and free them to spend more time on problems that requires their expertise.

As automated systems run it is typical that they can often perform many discrete steps and generate large amounts of data, which must be accurately recorded and tracked to ensure successful operation of the pipeline. However it presents a challenge to the end user who needs to be able to interpret the final results and understand how they were obtained.

In this article we present an overview of a project tracking system which is intended to help with the management and presentation of the data within automated pipelines. We describe the dbCCP4i system which provides generalised tools to store information about the individual steps or program runs performed within automated PX structure solution software, and to present this data in an intuitive manner to the end user. The technical details of the system are described along with a set of code fragments to demonstrate how it can be incorporated into existing Python or Tcl scripts, and two case studies are presented describing practical usage of the system, including incorporation of the tracking system into the current release of MrBUMP.

2. Overview of the dbCCP4i Project Tracking System

The architecture and components that comprise the dbCCP4i project tracking system has already been described in some detail in a previous CCP4 newsletter article [6], however it is still useful to give a brief overview. The key components are shown below in figure 1 and are described in the following sections.

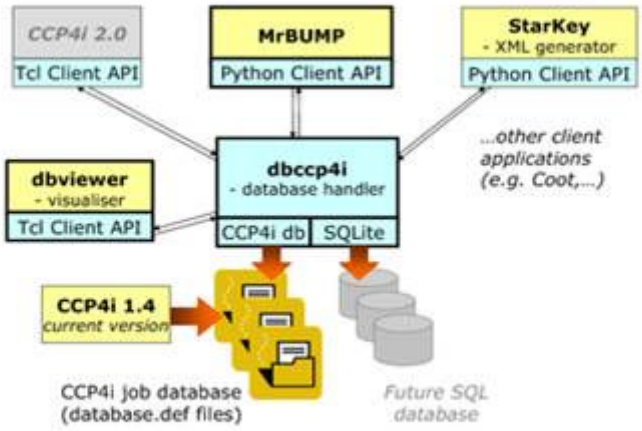


Figure 1: Architecture of project tracking system Project database handler

2.1 Project database handler

At the core of the system is the project database handler dbCCP4i, which is essentially a small server program that handles interactions between the database and other programs.

It processes requests from "client applications" (that is, an automated pipeline or any other program that wishes to store or retrieve data from the database) and interacts with the database on their behalf by performing storage and retrieval actions in response to their requests.

Using the handler hides much of the details of the database (described in the next section) from the client programs. It also allows multiple applications to access a single database simultaneously. The clients can be informed of changes to the database made by other programs via "broadcast messages" sent to all clients by the handler.

The handler is written in Python programming language, however the communications have been kept as language-neutral as possible by using a simple pseudo-XML markup to describe requests and responses, and by passing these between the handler and the clients via sockets. In practice, since the technical details of dealing with the socket communications can become quite complicated, "client API" libraries are provided to hide these details and simplify access to the database. Use of the client API libraries is described later on in section 3.

The system runs on both UNIX/Linux and MS Windows platforms.

2.2 Database for storing job data

The dbCCP4i system is built on the existing database used within the graphical user interface CCP4i [7], This records basic information about each program run (also called a "job"), including the date, input parameters, associated input and output files, and the outcome of the run.

The data itself is stored using the same "database.def" file infrastructure as CCP4i, and so is backwardly compatible with existing data already stored by users of CCP4i. This means that for example the visualisation tool described in the following sections can be used to look at old and new projects run by the user.

We now briefly describe how job record data is recorded within CCP4i, for those not already familiar with the system.

Within CCP4i users can organise their work into "projects". A project is simply a collection of data files plus an associated job database. Conventionally the data within a particular project will be associated with the same structure solution or part of a structure solution, however users or application programs are free to define projects in any way that they wish – there is no formally enforced definition of a project.

Within CCP4i and dbCCP4i, each of these projects is identified by a unique name or "alias" that is used as an identifier elsewhere in the system. The projects are implemented as a directory which contains the files in the project plus the database.def file which holds the raw job tracking data. The "job database" for each project contains job records, where a job is a run a software application or task (which itself can be a single program or a script that runs many programs in some sequence).

Each job record has the following information:

- An automatically assigned job id number
- A last modification date andtime
- A "task name" corresponding to the name of the program, task or script
- An associated status indicating success or failure
- An arbitrary user – or application-specified title
- The name of any assocated files, including: a log file, run file, and lists of input and output files

All these data items can be accessed via functions provided in the client APIs.

2.3 Visualisation of the job data using dbviewer

Users already familiar with CCP4i will be aware that the job database can be examined and interacted with via the chronological "job list" that is presented as part of CCP4i's main interface window. While this list-based view is useful, it also has some limitations and it can quickly become difficult to navigate the project history when a large number of jobs have been run, and also doesn't give any indication of how the jobs might be related to each other.

As part of the dbCCP4i system an alternative way of visualising the project history has been developed within the dbviewer program. Dbviewer presents the tracking data in an intuitive manner to the end user by showing the jobs within the project with links between them indicating the transfer of data. The user can interact with the view to see the files and other data associated with each job (see the figure 2).

The viewer is written in Tcl/Tk (for compatibility with CCP4i) and uses the Graphviz package [8] as an engine for generating the layouts of the history graphs that can be seen in the figure.

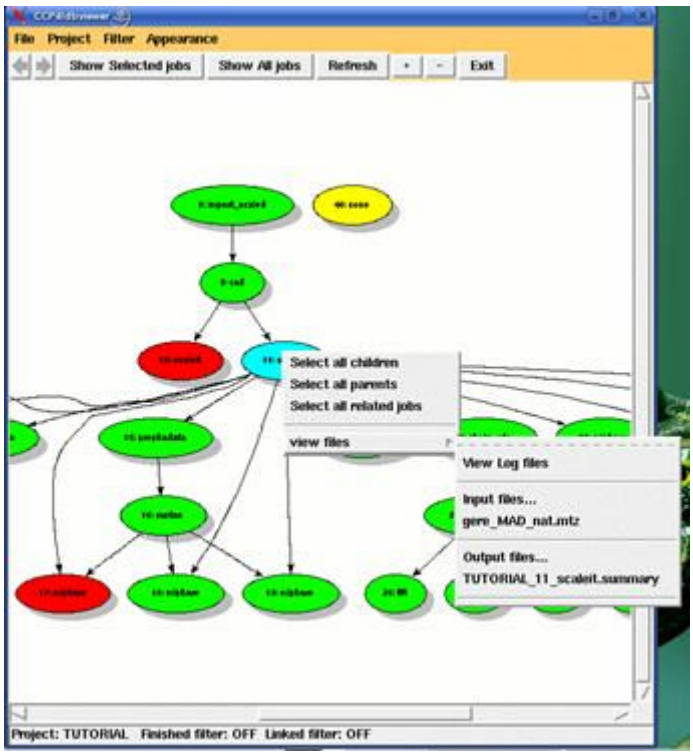


Figure 2: dbviewer: jobs with the links indicating the transfer of data files

Within dbviewer each job is represented by a "node" in the graph. Jobs are linked together by lines wherever an output file from one job is used as input to another job, to show the flow of data through the project. The colour of each node provides an indication of the status of the job, specifically green for completed jobs, red for failed jobs, and yellow for those which are still running.

The viewer also provides tools to interact with the job data as described in the following sections.

2.3.1 Selection tools

Dbviewer provides tools for users to select a subset of jobs to view, so that users can concentrate on the jobs that are of particular interest to them. For example, the visualiser can trace all the "antecedents" or "descendents" of a particular job selected by the user. Alternatively the user can manually select a subset of jobs to view. The user can also step back or forward through previous selections.

2.3.2 Filtering tools

The filtering tool allows users to change the view so that the unsuccessful or unconnected jobs are filtered out of the view.

2.3.3 Viewing the associated files

By right clicking on a job in the viewer, users can access a menu of associated files that are available to view, specifically the log file and the input and output files for the selected job.

2.3.4 Generate xml file for structure deposition

Using the selection tool, users are able to select the jobs which lead to the successful structure solution. Dbviewer then offers an option to write out the job information in an XML file which could be used for structure deposition. The XML file is generated using starKey program, which is discussed in a later section as an example of a client application which accesses the databases.

2.3.5 Real-time monitoring of progress of automated pipelines

Where the project tracking system has been incorporated into an automated program in order to store the jobs or steps that have been run within the pipeline, dbviewer can act as a monitor to watch the progress of the automated progress. As steps complete and their status changes, or new steps are added, dbviewer updates its display accordingly.

The case study integrating dbCCP4i the MrBUMP automated MR system is described in a later section and allows the dbviewer to be used in this way.

3. Using the project tracking system in PX software via the Client API libraries

Within dbCCP4i a set of "client API" libraries are provided which have been developed to make it easier for application programs to interact with the database. So far client APIs have been developed in Tcl and in Python. This section describes how the client API functions can be used, with the aid of example scripts in both Python and Tcl.

Before a client application program can access the database it must establish a connection to the database handler (example fragment #1 for Python programs, and #6 for Tcl scripts). When establishing the connection, the application can supply its name (shown as the text "ApplicationName" in these examples) – this is not currently used, but in future will be stored as part of any new records or other data added to the database by the application.

Once the connection has been established, the application can use it to access the data in the database – acquiring a list of the existing project names, opening an existing project, or create a new project (each shown in example fragments #2 and #7). (Note that it is possible for an application to have multiple projects open at any one time.) In the examples, the project name "myproject" is used.

After the application has opened a particular project (note that creating a new project automatically opens it), it can perform various operations on the job data in that project, as shown in examples #3 and #4 for Python and #8 and #9 for Tcl.

Examples of adding a new job to an open project is shown in examples #3 and #8. Jobs are referenced by unique integer id numbers within each project, so in each case the result of the operations shown should be a new id number for that project. This id number is then used when retrieving or modifying the data associated with that job – for example in #4 and #5, there are examples of setting the job title and status, and associating a program logfile and other input and output files.

Data can also be retrieved from the database. The examples in #4 and #9 show how to get a list of the jobs already defined in the database, how to retrieve job data such as title, status, and associating input and output files.

Finally, when the application has finished its interactions with the database, it should ideally close all the open projects and disconnect from the database handler, as shown in example fragments #5 and #10 for Python and Tcl respectively. In practice the system is robust enough that the handler can cope with unexpected disconnections by client applications (for example if the client crashes), still it is recommended to disconnect cleanly.

Python script:

1. How to initialise the connection to the handler

To talk to dbhandler, The following codes need to be included in the applications:

```
# import the client API
import dbClientAPI

# create a HandlerConnection instance, connect to the handler
conn = dbClientAPI.HandlerConnection('ApplicationName',True)
```

2. How to list the available projects & open/create a project

After connecting to the handler, you are ready to access the information in the database. For example, you want to know what the projects are available, do the following:

```
# list available projects
conn.ListProjects()
```

If you want to open an existing project, you need to give the project name.

```
# open an existing project
conn.OpenProject("myproject")
```

If you want to create a new project, give the project name and directory that the project is going to be hold.

```
# create a new project
conn.CreateProject("myproject","/home/project/myproject")
```

3. How to add a job

After you open an existing project or create a new project, you could add new jobs data or update the old jobs data. Adding a new job will return a jobid which you need to use for later updating or accessing the job data.

```
# add a new job
conn.NewJob("myproject")
```

4. How to modify and retrieve data for a job

The following codes show how to update or retrieve the job data.

```
# list all jobs in a project
conn.ListJobs("myproject")
```

```
# set job title, status etc
conn.SetTitle("myproject",jobid,"Refine mr coordinates")
conn.SetStatus("myproject",jobid,"FINISHED")

# add input/output files, logfile
conn.SetLogfile("myproject",jobid,"10_refmac5.log")
conn.AddInputFile("myproject",jobid,"abc.mtz")
conn.AddOutputFile("myproject",jobid,"molrep1_refmac1.pdb")

# retrieve input/output files
conn.ListInputFiles("myproject", jobid)
conn.ListOutputFiles("myproject", jobid)

# retrieve job title, status etc
conn.GetData("myproject",jobid,"TITLE")
conn.GetData("myproject",jobid,"STATUS")
```

5. How to finish up

After finishing accessing the database, you need to close all the projects that you opened and disconnect from the dbhandler.

```
# close project
conn.CloseProject("myproject")

# close connection
conn.HandlerDisconnect()
```

The following is the equivalent Tcl script:

Tcl script:

1. How to initialise the connection to the handler

```
# import the client API
source dbClientAPI.tcl

# Start the handler
DbStartHandler handlebroadcast

# Connect to the handler
DbHandlerConnect "ApplicationName" True
```

2. How to list the available projects & open/create a project

```
# list available projects
ListProjects

# open an existing project
OpenProject "myproject"

# create a new project
CreateProject "myproject" "/home/project/myproject" msg
```

3. How to add a job

```
# add a new job, return a job id
set jobid [NewJob "myproject" ]
```

4. How to modify and retrieve data for a job

```
# list all jobs in a project
ListJobs "myproject"

# set job title, status etc
SetData "myproject" jobid TITLE "Refine mr coordinates"
SetData "myproject" jobid STATUS "FINISHED"

# add input/output files, logfile
SetData "myproject" jobid LOGFILE "10_refmac5.log"
AddInputFile "myproject" jobid "abc.mtz"
AddOutputFile "myproject" jobid "molrep1_refmac1.pdb"

# retrieve input/output files
ListInputFiles "myproject" jobid
ListOutputFiles "myproject" jobid

# retrieve job title, status etc
GetData "myproject" jobid STATUS
GetData "myproject" jobid TITLE
```

5. How to finish up

```
# close project
CloseProject "myproject"

# close connection
DbHandlerDisconnect
```

4. Case Study

This section describes client applications that use dbCCP4i.

4.1 Case I: MrBUMP

MrBUMP is a framework that automates the determination of macromolecular structures via the Molecular Replacement method(MR). In MR, a trial model based on a known structure is used as a template for extracting the structural information of the target from the experimental reflection data. The success of the MR procedures is critically dependent upon the choice of trial model, and so the approach used in MrBUMP emphasises the generation of a variety of search models.

In good cases MrBUMP will give a single clear solution to an MR problem; in other cases it will suggest a number of likely search models that can be investigated manually by the scientists – typically around 10-15 search models, although it could be as much as ten times that. In preparation for molecular replacement each model structure also undergoes rounds of processing by various programs such as side-chain pruning based on sequence similarity between the target and template. The modifications are then carried out using various CCP4 programs such as Chainsaw or Molrep.

As a result the end user faces a challenge when reviewing and evaluating the results of the MrBUMP run, and in earlier releases end users needed to filter through many different output files and directories.

In order to address this problem, the most recent release of MrBUMP(0.4.0) uses dbCCP4i to record information from each of the steps involved in the MR process and the dbviewer is then used to present the data in a more intuitive graphical form. The graphical view makes it much easier for the end user to monitor the progress of the MrBUMP run, and to more easily find and examine a particular file associated with the processing of each search model. Thus the use of the dbCCP4i substantially improves the ease of use of MrBUMP for the end user.

It is important to note that MrBUMP uses the node-job representation in the graphical viewer in a slightly modified way: rather than representing specific CCP4 jobs, nodes in the viewer are used more loosely to represent stages in the course of the automatic pipeline whilst still providing access to the underlying log and data files. A typical run of MrBUMP breaks down into the following set of steps:

- Processing of the input target data.
- Carrying out the various searches (sequence, secondary structure, domain, multimer)
- Multiple alignment of target with search model sequences.
- Downloading search model PDB files.
- Preparing search models (side-chain prunings, superposition of structures, etc.)
- Molecular replacement.
- Refinement of molecular replacement results.
- Phase improvement.

Each of these steps can involve the use of several CCP4 or third-party programs and can occur multiple times depending on the number of search models used, so it makes more sense to represent the steps as nodes rather than the individual jobs. This helps to provide a more intuitive and informative view of what is happening in the pipeline. This is made possible by the large degree of flexibility provided by the dbCCP4i API.

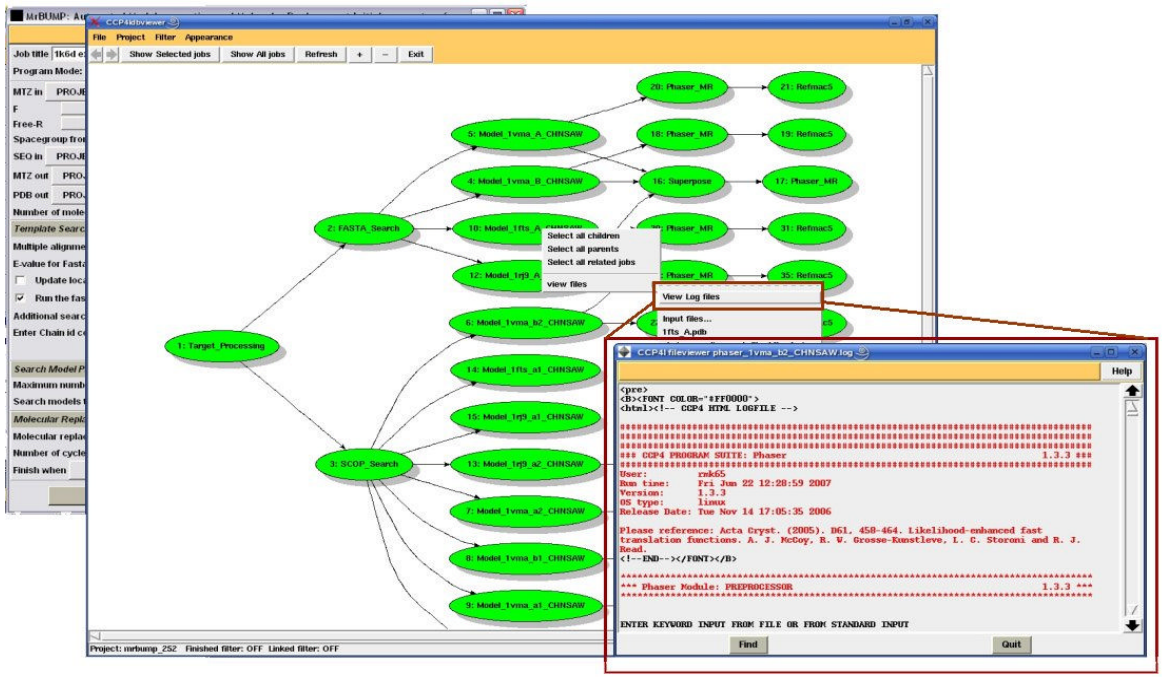


Figure 3: Example of the dbviewer showing the output from a MrBUMP job

4.2 Case II: starKey

Deposition of the structure with the wwPDB is an important part of the process of structure determination. Since dbCCP4i in principle allows diverse structure determination programs to access the same database, it should at some stage be possible for users to perform structure determination using a mixture of manual CCP4i tasks and automated pipeline programs and graphical model building tools, and have all the data history appear in the same project. It would then be possible at the end of the structure determination to collect together all the relevant data based on the job history in order to feed this directly into deposition.

To this end, starKey is a small program that has been developed as part of the dbCCP4i project, and which is able to collect the data history information from selecting the job nodes that lead to the final solution, and then using the "Generate XML" option previously described. This runs the starKey program with a project name and a list of jobs, which starKey then uses to extract the relevant data from the database in order to populate its XML file. In principle the resulting XML file could one day be used for structure deposition.

4.3 Case III: dbviewer

Dbviewer is another example of client application of the db handler. It is written in Tcl/Tk and talks to db handler via Tcl client APIs which described in section 3. The detail of dbviewer is described in section 2.3.

5. Availability

Version 0.2 of dbCCP4i is available for download from ftp://ftp.ccp4.ac.uk/bioxhit/dbccp4i-0.2.tar.gz

It includes the dbviewer and starKey applications as well as the core dbCCP4i handler and client APIs.

The system requires you to have CCP4 suite version 6.0 or newer, plus python 2.4 or newer and Graphviz package is required for the dbviewer. The dbviewer is compatible with the current CCP4i database, and can be run stand-alone.

MrBUMP version 0.4.0 incorporating dbCCP4i is available for download from www.ccp4.ac.uk/MrBUMP

This also includes the viewer and handles the installation of the various dbCCP4i dependencies.

6. Conclusions and Future Directions

The project tracking system is available for general CCP4 usage and could easily be used by other automation projects. The use of project tracking system means the outcome of an automation program is much clearer and accessible to the users. In addition, the Dbviewer can be used as a monitor to view the real-time progress of an automation pipeline.

There still remain some issues with the cohesion of MrBUMP's usage of dbCCP4i and the CCP4i interface but these are currently being worked upon. In particular, the issue of how to represent a MrBUMP job within the framework of a CCP4i project needs to be resolved. To address this, a MrBUMP job or any other automation pipeline will be represented as a sub-job or sub-project within a CCP4i project. When viewing the contents of a CCP4i project, a MrBUMP job will be represented by a single node in the graph. Double-clicking it will spawn a secondary viewer, detailing the contents of the MrBUMP job.

The dbCCP4i system is built on existing job database used in CCP4i. The system is compatible with current version of CCP4i. Thus CCP4i users can use dbviewer to look at the existing CCP4i projects as well as new projects created in CCP4i.

While the initial version of tracking database provides storage for job data which is useful for applications, one possible future direction is to combine this job history data with crystallographic data to produce a so-called "rich database". In particular, to provide a knowledge base which consists of the common crystallographic data items that are used in a software pipeline and can be shared between different applications. Currently we are working on a small version of the knowledge base with the intention of demonstrating the usage of the knowledge base in applications. The demonstration version of the knowledge base consists of dataset and heavy atom sub structure information which is used in the CCP4i "MLPHARE" task. An interface for storing and retrieving the data will be provided. We are also working on integrating dbCCP4i and dbviewer into CCP4i.

Finally, we are very keen to work with any automation software developers who would like to integrate dbCCP4i into their automation programs. Any feedback is very welcome.

7. Acknowledgements

This work is on dbCCP4i has largely been undertaken within the BIOXHIT project, which is funded by the European Commission within its FP6 Programme, under the thematic area "Life sciences, genomics and biotechnology for health", contract number LHSG-CT-2003-503420. Both the dbCCP4i and MrBUMP project have also been generously supported by STFC Daresbury Laboratory via the CCP4 project.

WY and PJB would also like to acknowledge the various contributors who have helped by providing many useful suggestions via discussions, emails and testing - in particular Ronan Keegan, Martyn Winn, Charles Ballard, Graeme Winter, Daniel Rolfe and Steven Ness.

8. References

[1] R.M.Keegan and M.D.Winn, *Acta Cryst*. **D63**, 447-457 (2007), "Automated search-model discovery and preparation for structure solution by molecular replacement"

http://www.ccp4.ac.uk/MrBUMP/

[2] S.Panjikar et al, *Acta Cryst*. **D61**, 449-457 (2005), "Auto-Rickshaw - An automated crystal structure determination platform as an efficient tool for the validation of an X-ray diffraction experiment"

http://www.embl-hamburg.de/Auto-Rickshaw/

[3] S.R.Ness et al, *Structure* **12**. 1753-1761 (2004), "Crank: New methods for automated macromolecular crystal structure solution"

http://www.bfsc.leidenuniv.nl/software/crank/

[4] HAPPy: P.Emsley, D.R.Rolfe and C.C.Ballard (unpublished) http://www.ccp4.ac.uk/HAPPy/

[5] G.Winter, *BSR 2007* poster, "Application of Expert Systems to MX Data Reduction";

CCP4 Newsletter 2007, "XIA2 – A brief user guide", (in this Issue)

http://www.ccp4.ac.uk/xia/

[6] P.J.Briggs and W.Yang, *CCP4 Newsletter* 45 (2007), "CCP4 in BIOXHIT: Managing and Visualising Project Data"

http://www.ccp4.ac.uk/newsletters/newsletter45/articles/ccp4_bioxhit.html

[7] E.Potterton et al*, Acta Cryst*. **D59**, 1131-1137 (2003), "A graphical user interface to the CCP4 program suite"

http://www.ccp4.ac.uk/ccp4i_main.php

[8] Graphviz: "Graph Visualization Software"

http://www.graphviz.org

New Features in Coot

Paul Emsley, Laboratory of Molecular Biophysics, University of Oxford, UK

SHELX Interface

A few enhancements have been made to the SHELX/Coot interface.

First, errors have been fixed in the way that Coot read SHELX .res files. The off-diagonal elements of the U matrix for anisotropic atoms are now read in the correct order. Also a pseudo-isotropic B-factor has been introduced (the average of the trace) which corresponds to the B factor in a SHELX output pdb file. Previously Coot used 10.0 as a placeholder here.

As a consequence, the Validation B factor variance plot has become useful - particularly so with SHELX's nicely refined high resolution atomic displacement parameters.

Coot now finds (large) gaps in the residue numbering and splits the chain on finding a such a gap. Thus waters are split into their own chain and hence water-specific analysis now works using .res files.

Secondly, Coot reads the SHELX listing (.lst) file and produces a GUI for it. Thus we are provided with a dialog box of buttons which describe and navigate to such things as potentially split atoms, bad DANGs or bumps.

Thirdly, Coot reads the output .fcf file from SHELXL - it does so by first converting the CIF file to an mmCIF file. The resulting structure factors are passed through the (relatively new) σ_a code in Clipper 2.0 to provide a σ_a map and σ_a difference map.

And lastly on the new SHELX functions, there is a convenience GUI interface which will read all the above files (.res, .lst, .fcf) given a single file-name.

NCS

Several enhancements have been made to the NCS handling in Coot. The copy residue range function in particular has been made more robust.

NCS skip

The '**O**'[1] key has been bound to the `skip-to-next-ncs-chain` function. When working on (or looking at) a particular residue in a chain that has NCS, one might think "what does the other chain look like here" - the '**O**' key provides a quick means to navigate to the corresponding residue other NCS related chains.

[1] think "Other Chain"

Non-default NCS operators

There are two scenarios here, depending on if the related structure has secondary structure or not.

For example, consider the case of a single chain composed of 2 domains, with a variable elbow angle between them and this single chain has 2 NCS copies in the asymmetric unit. It is quite possible that the different copies of the chain have different elbow angles and so the default operator which maps the whole copy on to the master molecule would not be optimal. It would better to have NCS copies for each domain separately. This is particularly important for NCS map overlays.

In this case, the recommended procedure is to use the Copy Fragment function[1] to create a new molecule that contains an atom selection that is just those parts of the asymmetric unit that are related by close NCS (e.g. the first 200 residues of a chain). Coot can then find the NCS in that partial structure and that can be used to generate an NCS operator to overlay the maps.

The other scenario is that the reference structure and NCS related copies are not protein and therefore do not have protein secondary structure[2].

In this case, Coot will not be able automatically determine the NCS operators. The solution is to provide a residue range and a pair of chain-ids to the function `manual-ncs-ghosts`. Coot will then use the residue range information to find the operator from a least-squares fit of one fragment onto the other (the automatic method is to use SSM to do this) and Coot is force-fed this operator. Coot will act as if it had generated the operator itself and proceed as normal.

Repo moved to Google Code

Coot now uses Google Code as the code repository:
http://code.google.com/p/coot/

Google Code has proved to be fast and reliable.

The current spec and schedule are available in the TODO file.

The mailing list has moved to JISCMail, this has relieved the system administrator in York of a burden, but the web representation of the threads leave something to be desired when compared to MHonArc or hypermail.

Structure as s-expressions

Using `(add-molecule molecule-expression molecule-name)` Coot can construct an internal representation of a molecule from a s-expression - a physical representation (i.e. a pdb file on a disk) is no longer required. The structure of the s-expression follows the mmdb coordinate hierarchy. Using an s-expression in this way, it is possible to send Coot structure data via a network connect without having to deal with files on a file-system. The

[1] `new-molecule-by-atom-selection`
[2] for example, RNA, DNA or ligands

application for this is from "driver programs", automaticed structure solution pipe-lines, refinement packages and the like that would like to show particular features to the user - perhaps even ask questions about them.

There is also a mechanism to update a given molecule using `(clear-and-update-molecule molecule-number molecule-expression)`. In this way it is possible to replace a given molecule with (for example) atoms in different places. This allows us to represent on-the-fly molecular dynamics, rotamers options, various loops or other partial models.

set-atom-attribute

The individual "atomic" attributes of atoms[1] are accessible to modification using the set-atom-attribute function.

The Active Residue

After discussions with Frank von Delft[2] the concept of "Active Residue" was added[3]. What that means is that Coot finds the residue[4] that is close to (or at) the centre of the screen and provides a residue specification for it that can be used in scripting functions.

Key Bindings

If you look at the console when you type a key letter, those without a binding result in something like

```
key: 110
```

when you press it. Those that do have a internal binding don't say anything[5].

Redefining `graphics-general-key-press-hook` is how you attach your bindings:

```
(define graphics-general-key-press-hook
  (lambda (key)
    (cond
     ; H key
     ((= key 120) ; X key
      (refine-active-residue))
     ((= key 104) ; H key
      (refine-active-residue-triple))
     ((= key 107) ; K key
      (auto-fit-rotamer-active-residue))
     ;; other binding (place atom at pointer when V key hit)
     ((= key 118) ; V key
      (set-pointer-atom-is-dummy 1)
      (place-atom-at-pointer)))))
```

[1] such as the x, y, z coordinates, the occupancy or alt-conf

[2] Frank's philosophy (if I may paraphrase) is "I want to get my Coot session over with as quickly as possible and I want to do it using the keyboard - making me use menu items is bad and slows me down" - I have some sympathy with this view.

[3] I'd been resisting the idea because I thought that it would be impossible or very difficult to port to Python, I needn't have worried as it turns out - Bernhard Lohkamp made it work

[4] actually, it's the atom

[5] Coot already has several key bindings, for example R,T,Y,A,S,I,D,L,C,E,Q,W,O,P,B,N,M,Esc,Ret

I hope that this is clear that this binds `(refine-active-residue)` to the 'X' key, `(refine-active-residue-triple)` to the "H' key, `(auto-fit-active-rotamer)` to the 'K' key and `(place-atom-at-pointer)` to the 'V' key. Quite useful functions[1] - especially 'X' and "K'.

Coot has many functions which take specifications for a residue or an atom - using key bindings and the results of `(active-residue)` bespoke short-cuts can be created that work on the residue at the centre of the screen[2].

Concluding remarks

I should note however, that fixing crashes, bugs and mis-feature oversights has been the highest priority. For the last 2 years, this has pretty much been the mainstay. I'd like this to change - which requires rather more discipline in the initial writing of the code on my part than I had currently been used to. An embedded testing architecture has been introduced - tests can be (and have been) written in Coot's scripting language, which makes testing portable and automated[3]. After writing a new feature I now think "How can I write tests for this?", rather than using the GUI one or twice to see that it vaguely works in the example case. I'm hoping that substantial use of this architecture will result in less reworking of functions and ultimately expedite Coot development. We'll see.

[1] Perhaps they should be the default.

[2] Note that the P key is bound to the "Update from Current Position" function, which takes you to the nearest displayed atom, (with a preference for CAs, if it can find one) this will also update the attributes in the Go To Atom window, so that "Next Residue" will take you to the next residue relative to the one at the current position.

[3] Peter Zwart and Ralf Grosse-Kuntsleve double-teamed me to add straws to my resistant humpy back

CCP4mg:The Picture Wizard

Liz Potterton and Stuart McNicholas

YSBL, University of York, York YO10 5YW

Email: ccp4mg@ysbl.york.ac.uk

1 Introduction

Version 1.0 of CCP4mg was released at the end of 2006 and by the time you read this CCP4mg 1.1 should be available. This article presents some key developments in the new version.

The main interface tool in CCP4mg is the **Display table** window which lists all of the data loaded into the program and for each data object lists the display objects which represent the data.

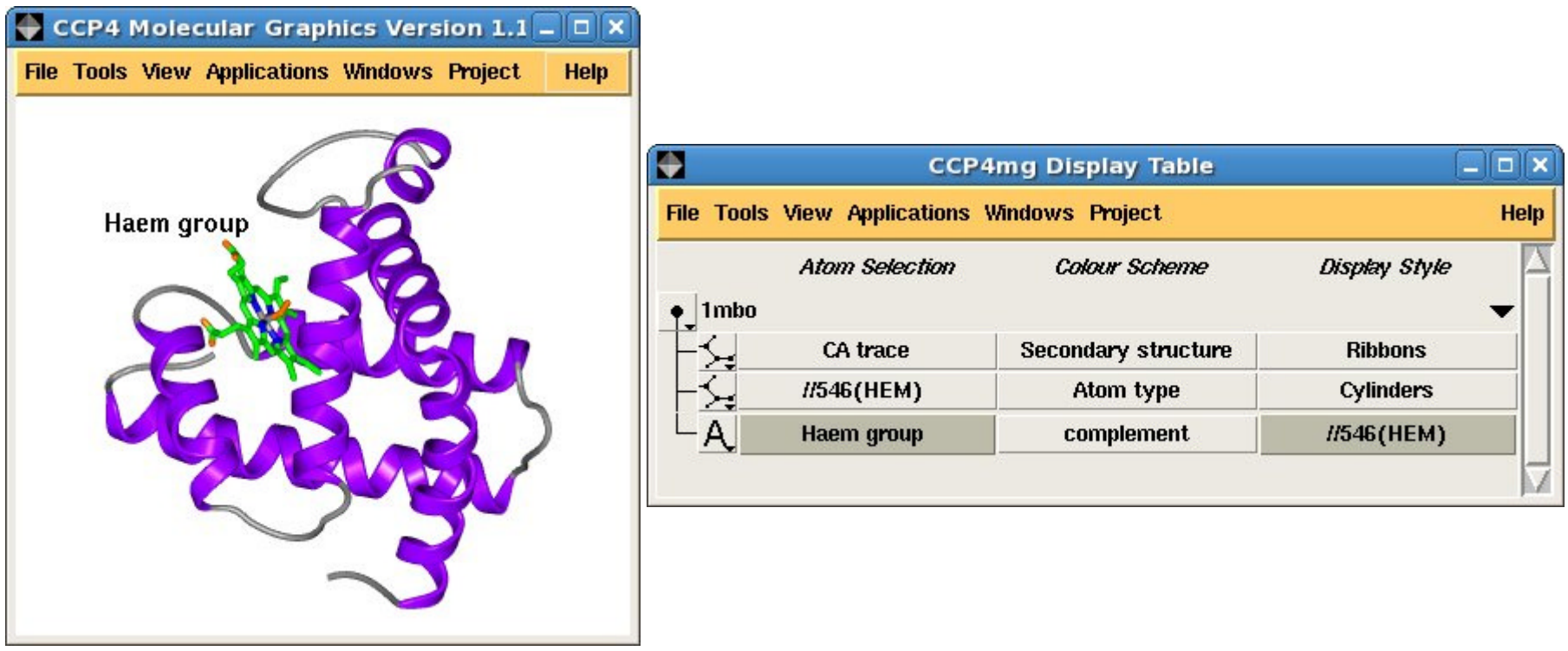


Figure 1. A simple picture of myoglobin (1mbo) and the corresponding display table

Figure 1. shows a screen image and the corresponding display table interface. The loaded PDB file, **1mbo** is shown in the display table and listed for it are three display objects:

- The **CA trace** is coloured by **Secondary structure** and displayed as **Ribbons**
- The haem group, **//546(HEM)**, is coloured by **Atom type** and displayed as **Cylinders**
- The annotation text "**Haem group**" has the colour that **complement**s the background colour and is placed by the haem group, **//546(HEM)** (offset by a small amount not shown on the interface)

Clicking on the various items in the display table opens menus and windows to enable changing the selected parameters. This interface is simple and powerful but initial setting up of a scene like this can take a little time. To save the user time in setting up some standard scenes we have developed the Picture Wizard.

2 The Picture Wizard Interface

The interface to the Picture Wizard shows thumbnail pictures of scenes grouped into *Style Folders* such as: *nucleic acid*, *ribbons* or *ligand binding site*. The interface to the nucleic acid style folder is shown in Figure 2. The user chooses one of the pictures (clicking a *Create picture* button) and their loaded model will be redrawn in the same style. There are options to retain any existing display objects (the default is not to) or to select limited fragments of the model to be redrawn in the new style so it is possible to build up a scene from several of the Picture Wizard templates.

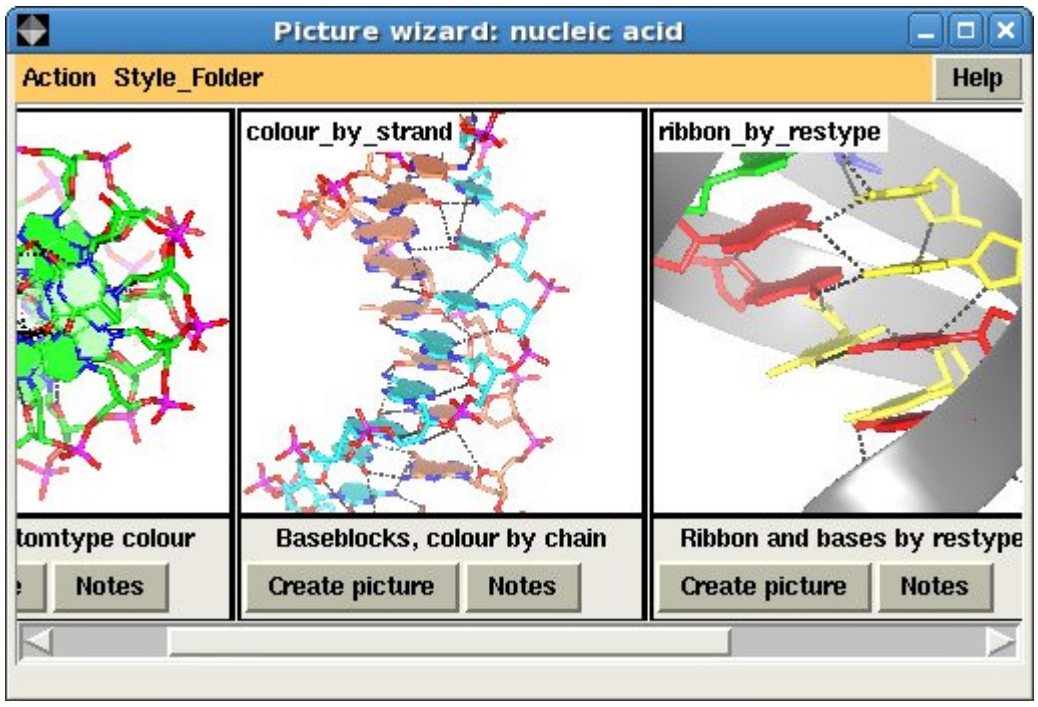


Figure 2. The Picture Wizard interface showing nucleic acid templates

The Picture Wizard also has tools to quickly add annotation to a scene. A right-mouse button click on an atom opens a pop-up menu with an option to **Add Annotation**. By default the annotation text is the residue or atom name of the picked atom in the user's preferred format; alternatively the user can enter some arbitrary text. The text is placed by the picked atom and its position can be adjusted interactively. This eliminates the need to add annotation to images in a separate drawing program.

The Picture Wizard functionality is also accessible from the coordinate file selection window. The file selection window has a menu listing the Picture Wizard templates and if a template is selected it will be applied to the coordinate file as it is loaded. For example a series of scenes created with myoglobin, 1mbo, are shown in Figure 3. and Figure 5.

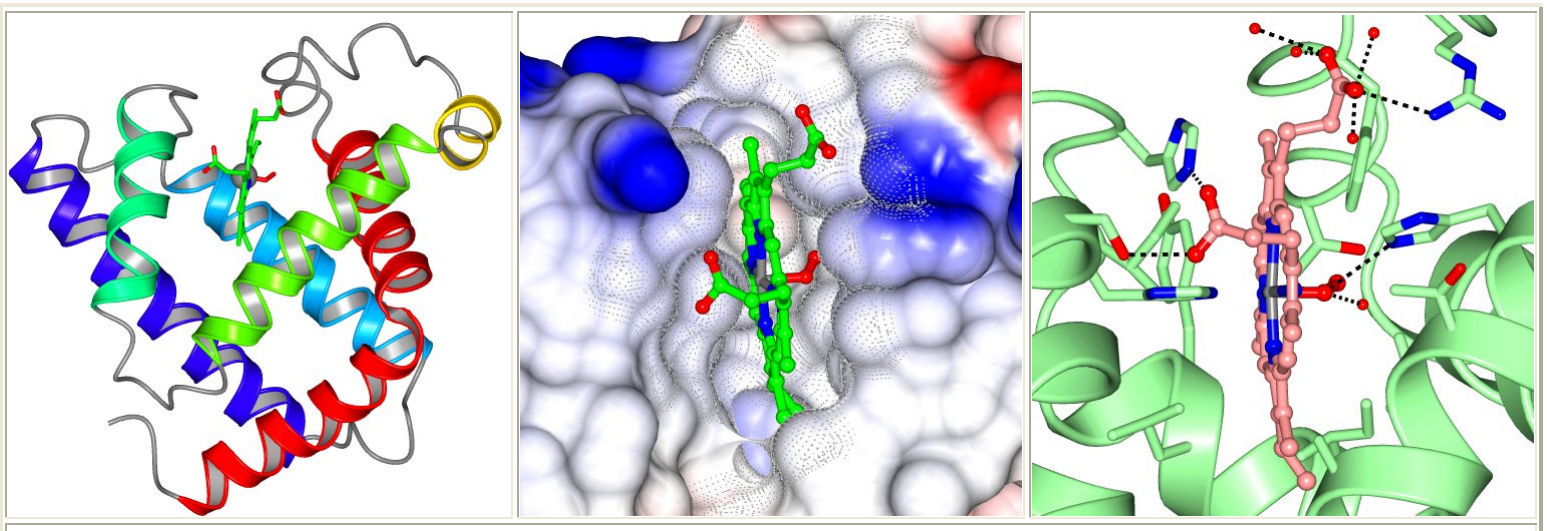


Figure 3. Myoglobin, 1mbo, scenes created with various Picture Wizard templates: ribbons coloured by blend thru model, ligand binding site surface and ligand binding site.

3 Picture Wizard Templates

The Picture Wizard template library has two files associated with each template: an image file for display in the interface and a template file which is used by the Picture Wizard application to create the appropriate display objects for the model. A user can create their own Picture Wizard templates either by modifying one of the distributed templates or from a scene that they have set up. The Picture Wizard interface has options to save both an image of the current scene and to create a template file based on the current display.

The templates distributed with CCP4mg are intended to be flexible and to handle a variety of model structures and so tend to be quite complex. Users creating their own templates may be happy with a simple template that can be applied to series of similar structures. You only need to be concerned with the template file format if you wish to customise it.

Template files contain several sections including a short title and documentation. The two important sections are the CHOICES section which defines a user interface such as shown in Figure 4. and the SCRIPT section which defines the display objects to be created. The user interface defined in the CHOICES section is presented to the user when they select a picture from the Picture Wizard interface and typically it requires the user to select which of the loaded models is to be redrawn and, optionally, to select particular chains, residue ranges or ligands. Additional options such as whether to label residues may also appear.

The SCRIPT section uses the Picture Definition File format (described below) to define the display objects that are to be created and drawn. The script has access to the parameters input to the CHOICES interface by the user and the script can query the loaded structure - for example to find out if the structure contains nucleic acid or solvent.

4 Picture Definition Files

Earlier versions of CCP4mg used a Python scripting language facility called Pickle to save the program status. The saved 'pickle' files have the extension *pkl* but are not human readable. The new Picture Definition File also completely saves the program status but is a more accessible format that can be written by other programs or edited by users that choose to edit scripts. The files are usually given the extension *mgpic.py*. The aspects of CCP4mg that are saved in a picture definition file are:

- the data objects and display objects that are shown in the Display Table
- user preferences
- the view
- possibly data from applications

The Picture Definition Language is based on the Python scripting language and uses the Python syntax for creating a series of objects[1] with specified attributes. Some examples of the object types:
MolData - model data object
MolDisp - model display object
Annotation - annotation display object attached to a MolData object
ParamsManager parameter manager - a group of preferences corresponding to one Preferences window
The object types and their attributes are documented (as part of built-in documentation and on the CCP4mg website) but probably the easiest way to 'read' and edit a picture definition file is to load it into CCP4mg and use the interface!

```
MolDisp ( selection = 'catrace',
          colour = 'secstr',
          style = 'SPLINE'    )

MolDisp ( selection_parameters =  {
                    'select' : 'nopeptide',
                    'monomers' : ['//546(HEM)' ]  },
          colour = 'atomtype',
          style = 'BALLSTICK'  )

Annotation ( text = 'Haem group',
             colour = 'complement',
             position = [-4.98, 9.94, -9.50, '//546(HEM)'],
             font =  {   'family' : 'helvetica',
                         'slant' : 'r',
                         'weight' : 'bold',
                         'size' : 14  } )
```

Figure 4. Example of Picture Definition File - the display objects that are shown in Figure 1.

The example in Figure 4 shows the definition of the three display objects that appear in Figure 1. For the two *MolDisp* objects (model display objects) the *selection*, *colour* and *style* attributes are specified. For the second *MolDisp* object specifying the selection requires two parameters and these are grouped as a *dictionary* (a Python language construct enclosed in curly brace {}) called *selection_parameters*. The third, *Annotation* object has attributes *text*, *colour*, *position* and *font*.

A Picture Wizard template file contains the definition of display objects similar to those shown in Figure 4 but this example highlights an important issue for template files. The name of the highlighted group, '//546(HEM' is given explicitly in this picture definition but in different structures the group to be highlighted will probably have a different name. To handle this the template can either ask the user to select a group (using the CHOICES interface) and/or can query the loaded structure to find the 'monomer' groups (i.e. not amino acid, nucleic acid, solvent or solute) and highlight them all.

5 Conclusion

The CCP4mg developers will be happy to hear any suggestions for further useful picture wizard templates or to advise if you are creating your own templates.

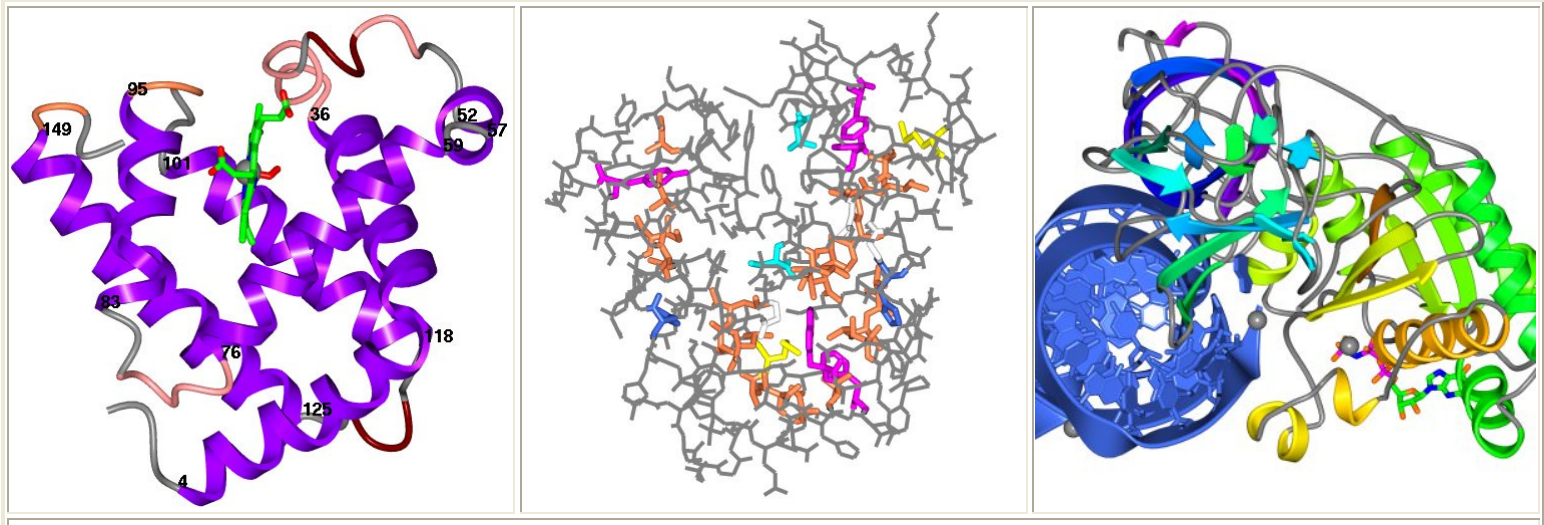


Figure 5. Myoglobin, 1mbo, scenes created with secondary structure and protein core Picture Wizard templates and the protein-RNA complex of 1b23 picture created with the ribbon colour blend template.

6 Useful Links

- The CCP4mg website is at www.ysbl.york.ac.uk/~ccp4mg
- Documentation for Picture Wizard
- Documentation for Picture Definition Files

Footnote

[1] We have used terms 'object' and 'type of object' rather than the programmer's terms 'instance of class' and 'class' in the hope that they are less intimidating to non-programmers.

Liz Potterton & Stuart McNicholas, July 15th 2007