Acta Crystallographica Section D
Biological Crystallography
ISSN 0907-4449

A graphical user interface to the *CCP*4 program suite

Elizabeth Potterton,[a]* Peter Briggs,[b] Maria Turkenburg[a] and Eleanor Dodson[a]

[a]Department of Chemistry, University of York, York YO10 5YW, England, and [b]Daresbury Laboratory, Warrington WA4 4AD, England

Correspondence e-mail: lizp@ysbl.york.ac.uk

Received 23 October 2002
Accepted 9 April 2003

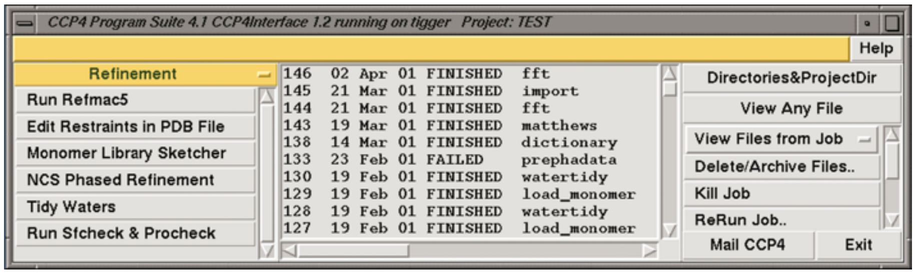
*CCP*4*i* is a graphical user interface that makes running programs from the *CCP*4 suite simpler and quicker. It is particularly directed at inexperienced users and tightly linked to introductory and scientific documentation. It also provides a simple project-management system and visualization tools. The system is readily extensible and not specific to *CCP*4 software.

1. Introduction

The Collaborative Computational Project, Number 4 (CCP4) is a collaboration in crystallographic software development that is best known for distributing a suite of crystallographic programs and software libraries (Collaborative Computational Project, Number 4, 1994). The software libraries support some standard crystallographic file formats and provide tools for command parsing, symmetry handling and other commonly used functionality. The programs distributed by CCP4 have been written in many different laboratories, but mostly use the software libraries and follow certain conventions to ensure a consistent style of keyworded input. CCP4 permanent staff ensure that distributed programs have consistent documentation and a common build mechanism which works for a wide range of platforms and provide user support.

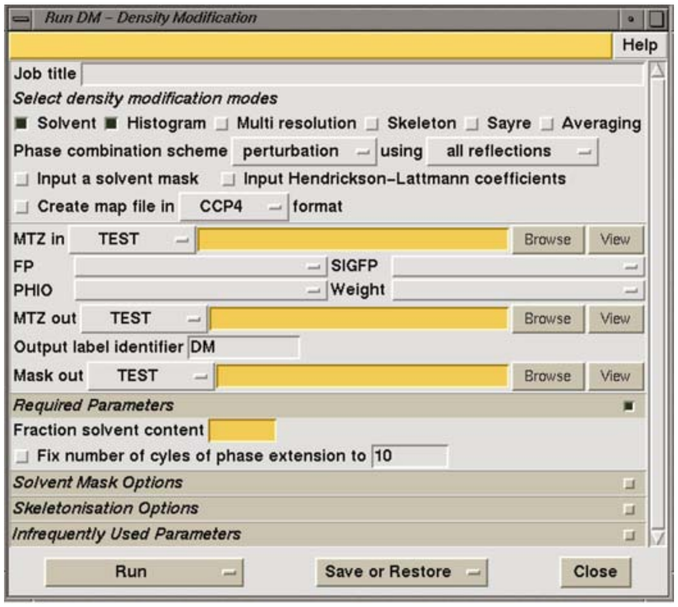
The *CCP*4 suite is a collection of stand-alone programs rather than one large program. For some operations, several programs are run in succession, with data transfer between programs being mediated by files in the standard formats supported by the software libraries. This approach allows flexibility for both programmers and users, but it can be difficult for inexperienced users. Most programs have a range of functionality that is controlled by keywords in a command script which is usually embedded in a Unix shell script. A Unix script might run several programs in succession and in itself encode some of the functionality. The user must have some programming skills to create these scripts or at least to edit parameters and file names in existing scripts. This process is slow and scripts are sometimes not up to date with the current best use of programs.

Running *CCP*4 programs has been much simplified by a graphical user interface called *CCP*4*i*, which provides standard scripts to run all of the common tasks and a graphical interface to each task that enables the user to enter parameters and names of input and output files. Each task interface is designed to require minimal user input to work in a default mode, but provides access to the full range of functionality available in the programs. In addition to interfacing to crystallographic tasks, *CCP*4*i* provides a project database that records details of each task run and can display a wide



```
146  02 Apr 01 FINISHED  fft
145  21 Mar 01 FINISHED  import
144  21 Mar 01 FINISHED  fft
143  19 Mar 01 FINISHED  matthews
138  14 Mar 01 FINISHED  dictionary
133  23 Feb 01 FAILED    prephadata
130  19 Feb 01 FINISHED  watertidy
129  19 Feb 01 FINISHED  load_monomer
128  19 Feb 01 FINISHED  watertidy
127  19 Feb 01 FINISHED  load_monomer
```

Figure 1
The layout of main window of the *CCP4i* graphical interface. The left-hand frame of the window is the task menu, the centre frame is the Job List and the right-hand frame contains utilities.



Job title
Select density modification modes
■ Solvent ■ Histogram ☐ Multi resolution ☐ Skeleton ☐ Sayre ☐ Averaging
Phase combination scheme perturbation using all reflections
☐ Input a solvent mask ☐ Input Hendrickson–Lattmann coefficients
☐ Create map file in CCP4 format
MTZ in TEST
FP SIGFP
PHIO Weight
MTZ out TEST
Output label identifier DM
Mask out TEST
Required Parameters
Fraction solvent content
☐ Fix number of cyles of phase extension to 10
Solvent Mask Options
Skeletonisation Options
Infrequently Used Parameters

Figure 2
An example of the layout of a task interface: the *DM* task interface. The window is split vertically into frames. The top frame is the protocol frame for key decisions and the next frame is for file selection. All subsequent frames have a title bar with the name of the frame. Frames containing less popular options are closed by default but can be opened by clicking on the title line.

range of file formats. There is extensive documentation and there are tutorials that teach the use of the programs and some basic crystallography.

The interface is written in the Tcl scripting language using the Tk graphical toolkit (Tcl Developers; http://www.tcl.tk). The BLT extension to Tk is used to create graphs and tables. These packages, and therefore *CCP4i*, will run on many operating systems, but those particularly relevant to crystallography are Unix, Linux, Windows and OSX.

2. Description of the graphical user interface

CCP4i consists of a core system and a series of 'tasks'. The core system manages the project database and has tools for file visualization. A task is the interface to a script which may run one or more programs. Each run of one task is referred to as a 'job'.

The graphical interface of *CCP4i* consists of a main window, shown in Fig. 1. On the left-hand side of the main window is a menu listing the tasks; when a task is selected, a separate task interface window is opened. In the centre of the main window is the 'Job List', which lists all of the jobs which have been run for the current project. The user can select a job from the list to apply a utility such as displaying input or output files from the job or rerunning the job.

Each task within the interface has a separate interface window which presents the user with the options for running the task, allows them to select input and output files and to start a job. All task windows have a similar layout to the interface shown in Fig. 2 for the *DM* density-modification program (Cowtan & Main, 1998). The task interface is divided into horizontal folders. The top folder, called the 'protocol folder', has the most significant options that determine the action of the task. Next is the 'file folder', in which the user can select input and output files and also select the column data from an input reflection data file in MTZ format. Each subsequent folder contains groups of related options whose functionality is indicated by the name in the title line of the folder. All folders except the protocol and the files folder may be closed so that the body of the folder is not visible. The title line of a folder is always visible and clicking on this line will open or close the folder. The folders are arranged with the more significant groups of options nearer the top of the window; folders containing infrequently used options are closed by default.

An important feature of the task interface is that as the user selects options, the interface is adjusted to display only the appropriate sub-options. In particular, the rest of the interface may be customized depending on choices made in the protocol folder.

The default parameters for a task will perform the most popular function of the task and the user should only need to select files and sometimes enter some non-defaultable data such as the expected solvent content. There are two mechanisms for providing additional information on each parameter in the window: the message line at the top of a window gives one line of information dependent on the current cursor position and a mouse click on any parameter brings up a web browser to access documentation for that parameter. There is also introductory documentation for each task. The documentation is distributed with the *CCP*4 release and is also available on the CCP4 web site.

3. Implementation of tasks

The interface mechanism is not limited to *CCP*4 programs and writing a new task interface is straightforward. The application programmer's interface (API) functions and the process for creating a new task interface are well documented. The documentation and code for a simple demonstration task is available at http://www.ccp4.ac.uk/ccp4i/developers.html. Creating a task interface does not require any changes to the program.

Each *CCP4i* task interface requires three files to define the task.

(i) A def file that contains a list of all of the parameters for the task. For each parameter, the name, data type and default value must be provided.

(ii) A GUI script that defines the appearance of the task window.

(iii) A run script that executes the program(s) and that is spawned as a separate process from the main *CCP4i* graphical interface.

The GUI and run scripts are written in Tcl, but for straightforward cases they can be written almost entirely using the *CCP4i* API, which is a library of simple high-level procedures. For less generic tasks, all of the functionality of the Tcl scripting language and if necessary the Tk graphical toolkit can be used to create task-specific functionality.

3.1. Data types

The data type of each parameter specified in the def file determines the type of widget used to represent a parameter in the interface. For example: a parameter of type `_logical` (*i.e.* a Boolean with allowed values 'true' and 'false') will be represented in the graphical interface by a radiobutton, which can be on or off. A parameter with the type `_positiveint` (a positive integer) will be represented by an entry field into which the user can type a number. The data type of any parameter also determines the permissible values for that parameter; if the user enters inappropriate data they will be warned by the input widget changing colour. Examples of inappropriate data would be a numerical value outside the specified allowed range or the name of a non-existent file for an input file. Common data types are defined in a separate file, but there are APIs to enable the task-interface programmer to define new data types. It is commonly necessary to create new data types for parameters that are rendered as multiple-choice pop-up menus. For each pop-up menu, there is a data type whose key attribute is a list of the options that are to appear on the menu.

3.2. The GUI script

The GUI script defines the appearance of the graphical interface to a task. The script uses a library of high-level graphical functions that are written using the Tk graphical toolkit. The graphical interface is defined one line at a time by a call to one of the library functions that each draw one line of interface. The most generic of these functions is the `CreateLine` function, which will draw a line containing any combination of text labels, entry widgets, pop-up menus and buttons. There are also specific library functions to create the file-selection and MTZ-column-selection lines (the latter are discussed in more detail in §3.3). The input to these library functions also contains a message-line help text and links to additional help files. The use of the high-level graphics library dramatically reduces the number of lines of code required to define the interface and helps to create task interfaces of consistent appearance.

A key feature of the graphical interfaces is that they are dynamically customized so that the user only sees the sub-options relevant for the options that they have already chosen. In order to support this, it is possible to control the visibility of each line in the interface: if the content of a line is not relevant for a selected option, then that line is made invisible. Conditional visibility is implemented *via* an optional argument to each line-drawing function that specifies a parameter that controls the visibility of the line and value(s) of this parameter for which the line should be invisible. Several lines can be grouped together into a 'frame' for which the visibility is controlled by one parameter. The lines and frames are grouped into the folders, which are clearly delineated in the task interface. The visibility of each folder can also be made dependent on the value of a parameter. Typically, a parameter that controls visibility of a line, frame or folder will have a limited number of valid values that are presented to the user in the form of a pop-up menu. When the user selects a new option from the menu, the value of the parameter is updated correspondingly and the visibility of any lines, frames or folders controlled by this parameter is updated automatically.

3.3. File selection

The graphical interface to all file selections is provided by the functions `CreateInputFileLine` and `CreateOutputFileLine`. Each of these functions creates one line in the graphical interface. All file-selection lines have a consistent appearance and contain (i) a pop-up menu that lists the defined projects and directory aliases to enable rapid navigation to commonly used directories, (ii) an entry field for the file name, (iii) a 'Browse' button that will bring up a more sophisticated file-selection window and (iv) a 'View' button that will display the selected file. Sometimes, when the user selects a file in some standard format it may be read to extract information that is used to set default values for certain parameters in the task interface.

Handling of MTZ files, which store the reflection data, requires some additional interface tools. Files in this format may contain an arbitrary number of columns of data. Associated with each column is an indicator of the data type and a unique label. Examples of the data types are intensities, structure-factor amplitudes, standard deviations, phases, figures of merit and free *R* flags. It is usually necessary to select the appropriate data columns for input to any program. The interface simplifies this selection by presenting the user with a menu listing the data-column labels for all data columns of the appropriate type in a selected MTZ file.

3.4. Run scripts

The run scripts that actually run the programs are generally written in Tcl, but since a script is run as a separate process it could be implemented in an alternative scripting language. Some scripts are very short and simple, running a single program, but others are quite complex, running several programs and including significant aspects of the task's functionality in the script. The run script can make use of an API library, which provides functionality to simplify creating command scripts, handling program errors and communicating with the main *CCP4i* process.

Most *CCP*4 programs require keyword command input. There are some conventions for the format of the command input, but this is not a very tight specification and it is not followed by all *CCP*4 programs or by the non-*CCP*4 programs that have also been interfaced. Generating the appropriate command input for the options and parameters selected by the user can be a complex task. This has been automated by a library function, `CreateComScript`, that will create the correctly formated keyword command input based upon a template file for the appropriate program. The `CreateComScript` function substitutes the actual parameter values for the parameter names that appear in the template file and writes the result to a program command file. Because of the complex nature of much of the programs' input, the template file also contains processor directives that correspond to the common programming command structures: `if-else-endif` statements, `case` statements and `for` loops. There is also a processor directive for generating the commonly used LABIN and LABOUT keywords that select the data columns in MTZ files.

The template-file mechanism is sufficiently flexible that generally only one template file is needed for each program, even if the program is used by different tasks. This mechanism has been able to handle the input to a wide variety of programs with very different command syntaxes, but the complexity of the template file is proportional to the complexity of the program input.

Another function used in the run script is `Execute`. It executes a program but provides additional functionality to create a master log file for the task, to handle program failure cleanly and consistently and to allow a user to view and optionally edit the command file before running the program. The latter option can be useful for expert programmers who may wish to modify the program input or test new tasks.

3.5. Example of code

Fig. 3 shows the line that appears in the *DM* task interface and that enables the user to enter the fractional solvent content of a unit cell. In order to implement this small part of the interface, the following lines appeared in the various files. The def file, `dm.def`, that defines the parameters, has

```
SOLVENT_FRAC  _fraction  "".
```

The parameter `SOLVENT_FRAC` is defined as having the type `_fraction`, which is defined elsewhere to mean a real number

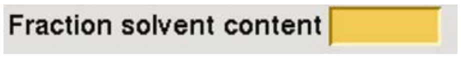


Figure 3
One line of the *DM* task interface.

in the range 0.0 to 1.0, and an initial value, which is an empty string. In the GUI file `dm.tcl`, the line shown in Fig. 3 is created by a call to the `CreateLine` function:

```
CreateLine line \
      message "Fraction of unit cell which is
        solvent (SOLC)" \
      help solc \
      label "Fraction solvent content" \
      widget SOLVENT_FRAC -oblig.
```

The first argument to this function, `line`, is returned and will contain an id code for the line drawn. The remaining arguments to the function consist of pairs of keywords and values. The keyword `message` is followed by the text that appears in the message line if the cursor hovers over the widget. The keyword `help` is followed by the name of a target in an html file; this will be used to link directly to the appropriate help text for this parameter. The keyword `label` is followed by the text seen in the window and the `widget` keyword is followed by the parameter name. The type of the parameter, defined in the `dm.def` file, determines the type of widget to be drawn. In this case, it is an entry widget into which the user can type a value. The final argument, `-oblig`, indicates that the parameter is obligatory for the task to run. This fact is indicated to the user by the gold contrast colour of the entry field; this will turn to white after the user has entered a value in the range 0.0 to 1.0.

Finally, in order to convert the value entered by the user into a command recognized by the *DM* program, the `CreateComScript` function will use a command template file containing the line

```
1 solc $SOLVENT_FRAC.
```

The initial parameter on each line of the template file is a logical variable which indicates whether this line is to be interpreted or ignored. The value of 1, or 'true', in this example means this line will always be interpreted. For some lines, this parameter may be the name of a variable or a short script which needs to be evaluated. The text `solc` is output to the program command file as it is, but the parameter `$SOLVENT_FRAC` is replaced by the value entered by the user.

3.6. Running jobs

Each run of a task is referred to as a job. Every job is run as a separate non-graphical process controlled by a Tcl script called *ccp4ish*. To run a job, *CCP4i* first writes all the necessary information to run the job to a def file and then starts a new *ccp4ish* process. The *ccp4ish* process reads the def file and then starts the appropriate run script for the task. *ccp4ish* includes a library of API functions including the `CreateComScript` and `Execute` functions described above. Another important function is to communicate with the main graphical interface using the Tcl interface to sockets. The main purpose of communication is to send progress reports to the main

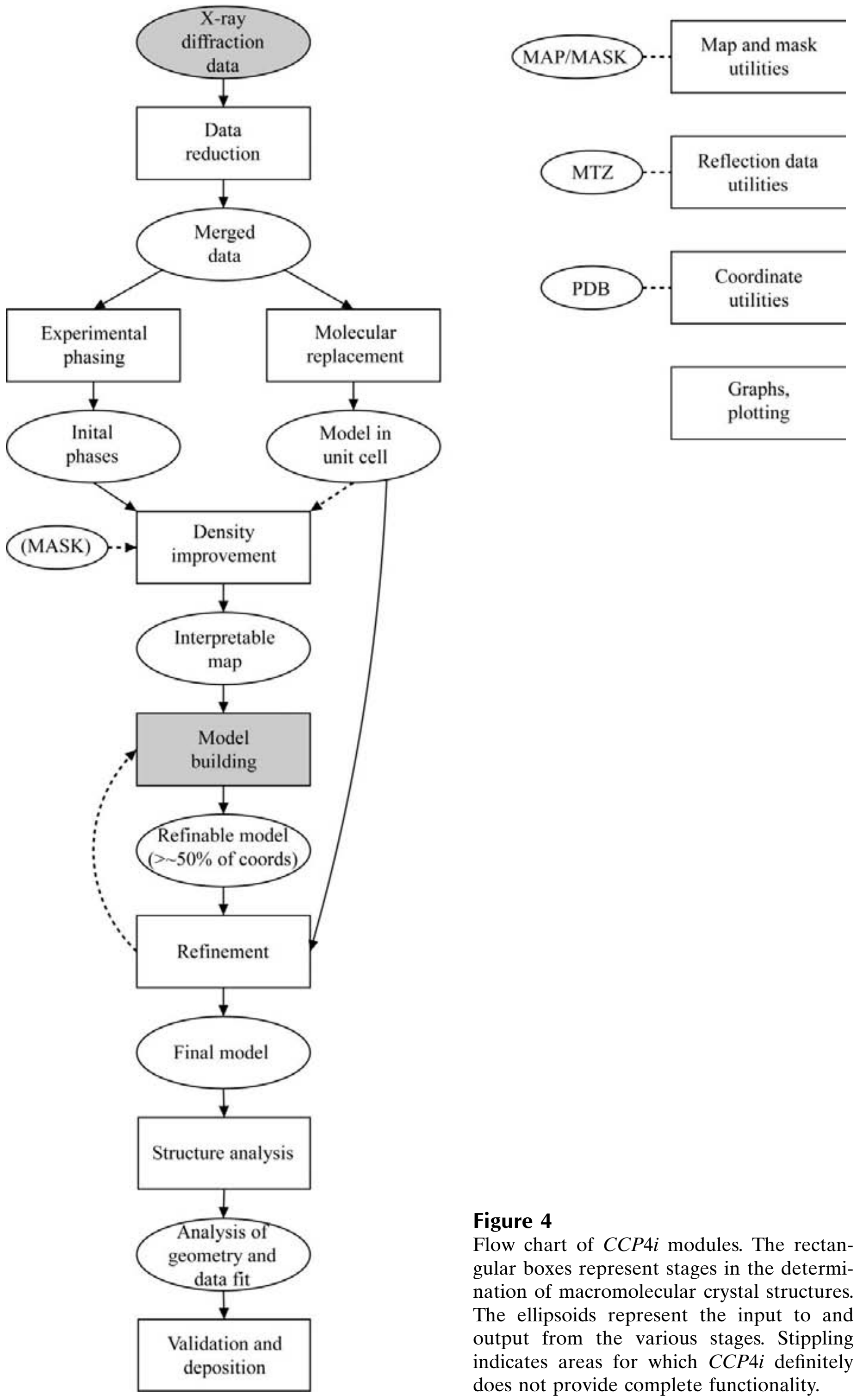


Figure 4
Flow chart of *CCP4i* modules. The rectangular boxes represent stages in the determination of macromolecular crystal structures. The ellipsoids represent the input to and output from the various stages. Stippling indicates areas for which *CCP4i* definitely does not provide complete functionality.

graphical interface and to report the names of any additional output files that the script creates.

Jobs can be run on the same machine as the graphical interface or, for Unix systems supporting the `rsh` command, on a machine in the local cluster. Jobs can be submitted to a batch queue, but *CCP4i* may need some customization to fit with the batch system of the installation.

4. Examples of structure determination functionality

The *CCP4i* tasks are grouped into modules. There is one module for most stages in the crystallographic process and utility modules with tasks for handling reflection data, coordinate data and map and mask data. *CCP4i* provides access to all the functionality required for a reasonably straightforward structure solution and refinement starting from processed data, but excluding the graphical model building. A flow-chart of the *CCP4i* modules is shown in Fig. 4.

The amount of functionality in a run script varies greatly. Some scripts just run one program, other scripts run one main program and include some (usually optional) utilities to perform things such as file-format conversion or creation of a map in the user's preferred format. Some of the run scripts closely follow existing Unix shell scripts such as the *Uniqueify* script, which ensures that an MTZ file had a complete set of unique reflections and contains a set of free *R* flags, or the *Solomon* script, which runs the *SOLOMON* density-modification program (Abrahams & Leslie, 1996) and other support programs. Other run scripts significantly extend the functionality of the programs, for example the *AMoRe* script and the *NCS-Phased Refinement* script.

The *NCS-Phased Refinement* script can be used in the early stages of refining a model derived from molecular replacement if there are two or more molecules related by non-crystallographic symmetry in the asymmetric unit. Initial phases are derived by the *DM* program from a mask that covers the NCS-related molecules. These initial phases are input to the *REFMAC* refinement program. The process is repeated for several cycles with a new mask and phases calculated at each cycle. The power of the method comes from the constraint on the phases from the NCS relationships (Pannu *et al.*, 1998; http://xplor.csb.yale.edu/xplor/xplor-faq/Q.247.html) and has been used successfully in the early stages of refinement of several structures (Roper *et al.*, 2000; Moroz *et al.*, 2001). The *CCP4i* script automates the running of mask generation, *DM* and *REFMAC*.

The *AMoRe* molecular-replacement program (Navaza, 1994) is usually run in a number of stages: data preparation, rotation function, translation function and fitting. The *CCP4i* script automates the process and runs all stages sequentially using 'mr' files (described below) to tranfer data between different runs of the program. Alternatively, individual stages of the process can be run separately with the input coming from mr files that the user can select and optionally edit. The mr files are an example of the approach used to simplify and automate transfer of data between different programs while allowing user intervention. An mr file used in the molecular-replacement module contains information about the rotations and translations that need to be applied to a specified initial model to give the molecular-replacement solution. *CCP4i* generates this file from the output file of *AMoRe* and can use it as input to further runs of *AMoRe* or to generate a PDB file representing the solution.

Similarly, *CCP4i* generates an 'ha' (heavy-atom) file as output from the heavy-atom search tasks in the experimental phasing module [currently interfaces to *PEAKMAX*, *RANTAN* (Yao, 1983), *RSPS* (Knight, 2000) and the non-*CCP*4 package *SHELX* (Sheldrick, 1998)]. The ha file contains information on putative heavy-atom solutions: the atom type, coordinates, temperature factors, occupancies and anomalous occupancies. The ha file can be used as input for the task interface to the heavy-atom refinement program *MLPHARE* (Otwinowski, 1991).

CCP4i will display the ha and mr files and has simple tools for the user to edit these files: particularly to 'comment out' some solutions so that they will be ignored if the file is used as input to another task. The ha and mr files and the editing facilities go some way to automating the structure solution while still allowing user intervention. They may also be a starting point for better communication between various software packages. The EU-funded AutoStruct project (http://www.autostruct.org) is considering using an improved ha file for communication between several software packages and the *CCP4i* can be modified to use any new format.

5. Project management

CCP4i is based around the idea of projects, where one project is expected to correspond to one crystal structure solution and all files relating to that project are expected to be in one directory referred to as the project directory. The user is not required to follow this organization, but it is recommended. There is always one current open project but the user can move quickly between projects by selecting from a list of all projects.

For every project, there is a project database that records every job run. The information that is currently saved in the project database for each job is the task name, the completion date and time, the status (usually 'running', 'finished' or 'failed') and the names of input and output files, including the log file. In the graphical interface for each task the user has the option to enter a title, which should be a short distinguishing description of the job, and this is also saved in the project database. A def file, which contains a full list of the input parameters for the job, is also saved.

All of the jobs run within the project are listed in the main window of the interface with the name of the task, time (if run on the present day) or date, status and title. There is a menu of utilities beside the Job List; any utility can be applied to any job selected from the list. The utilities are viewing input or output files of a job, killing a job that is still running, deleting output files of a job and rerunning a job (with the option of changing the input parameters).

This simple project-management system makes it easier for a user to recall the crystallographic process, to review results and if necessary to backtrack. There are two additional tools to aid project management: information about jobs run external to *CCP4i* can be entered and saved in the database and there is a notebook utility for users to enter comments on any of the jobs. Another benefit from using projects is faster file selection: by default, input and output files are expected to be located in the current project directory. *CCP4i* also has the concept of directory aliases; the user can give a short alias for any frequently used directory. Wherever the user needs to select a file, there is a pop-up menu that lists all projects and directory aliases so that the user can navigate quickly to the required directory.

CCP4i creates some additional files to manage the project: the database file called `database.def` and the def files containing the input parameters for each job. These files are saved to a subdirectory of the project directory called CCP4_DATABASE. It is expected that the project database will be developed in the future to record more information automatically and to use this information appropriately in subsequent tasks.

There are initiatives by the macromolecule database submission sites to automate the harvesting of data that are required for structure submission from the programs that have ready access to that data. This process is independent of any automation of the structure-solution process. *CCP4i* provides a user interface to the harvesting mechanism.

6. File visualization

All data files used throughout the structure solution and refinement can be displayed by the interface; for some file formats, *CCP4i* will start another appropriate display program. There are several means of accessing the file display: the file-selection lines and file-selection window have 'View' buttons and there is a utility associated with the Job List to view any input or output file for a selected job. It is straightforward to define appropriate viewers for additional file formats or add alternative viewers for formats that are already supported.

*CCP*4 program log files are in the process of conversion from simple text format to html format. Use of html format enables cross-linking within the log file and linking to external documentation. The interface will recognize the format of the log file and display it in an external web browser or in an internal text-display window.

Log files may also contain tabulated data, which is best visualized as graphs. The *loggraph* application can display the graphs defined in the log files. There is a standard format for a log file table that includes annotation to label the graphs within *loggraph*; each column of the data has a title which is used to label the graph. *loggraph* will also read any file containing simple unannotated tabulated data. The *loggraph* application has options to modify the appearance of the graph by changing colours or linestyles, editing title, axes titles or labels or adding annotation. The graph can be saved in PostScript format.

Two of the standard file formats, MTZ reflection data file and *CCP*4 map files, are binary files that cannot be visualized by any of the standard means. *CCP*4*i* will display the information stored in the header of these files. A graphical interface to the *SFTOOLS* program (Bart Hazes, unpublished work), provides additional means to inspect and manipulate the MTZ file.

Map files can also be viewed using the *Mapslicer* application, which displays two-dimensional sections of maps. The maps can be sectioned down any axis and *Mapslicer* can select Harker sections automatically. The application uses the Tk canvas widget, which is a simple and effective means of generating two-dimensional graphics. The map contouring is performed by an existing contouring algorithm written in Fortran and linked into a Tcl interpreter to appear as an additional command in the Tcl scripting language. By this means, more computationally intensive algorithms such as map contouring can be written in a more appropriate language but tightly linked with the scripting language part of the application.

The Tk canvas widget has also been used to display the three-dimensional molecular structure of small molecules in the *Monomer Library Sketcher* application. This is an interface to the LibCheck program that manages the library of geometric restraints for monomers used in restrained refinement (Vagin *et al.*, 2003). The *Sketcher* can be regarded as a visualiser for the geometric restraint library entries and it provides tools to create a new entry or edit an existing one.

7. Discussion

*CCP*4*i* has been a significant step forward in the usability of *CCP*4 software: it has made access to *CCP*4 programs considerably easier for inexperienced users and significantly quicker even for experienced users. The resultant software system is not a black box: the experienced user can see exactly what is happening and control the fine details of program input. All output files and intermediate data are accessible. Because all files are accessible, a user can alternate between using *CCP*4*i* and running programs by some other means. A potential pitfall of this approach is that a user could, for example, move files and invalidate the information in the database. In practice, however, crystallographers tend to understand the system well enough not to do this. A reservation sometimes raised about this system is that it encourages inexperienced crystallographers to treat the process as a ‘black box’ and pay insufficient attention to details of program input and to program output. However, this possibility has always been there and the *CCP*4*i* project database does at least enable users to review earlier steps and backtrack when problems arise.

*CCP*4*i* is readily extensible, implementation of individual tasks is straightforward and there is no technical reason that prevents interfacing to non-*CCP*4 software.

*CCP*4*i* will serve as a platform for future developments to improve the scientific software and to increase automation. The use of standard scripts, which the user does not need to interact with directly, makes it possible to develop both the programs and the scripts in a way that is transparent to the user. The project database maintains a record of the structure-solution and refinement process. The database can be developed in the future to record more of the results of each job and this information could be used in decision making in a more automated process. *CCP*4*i* will interface to the tools that automate harvesting the experimental information required for submission of structures to the structural data banks.

Our thanks to Darren Spruce for allowing us to use his original version of the *loggraph* program, Leo Caves for code used in the Monomer Library Sketcher and Alun Ashton for porting to Windows. Many thanks to many programmers and users for advice and feedback, particularly Phil Evans, Garib Murshudov, Heidi Schubert and Johan Turkenburg.

References

Abrahams, J. P. & Leslie, A. G. W. (1996). *Acta Cryst.* D**52**, 30–42.

Collaborative Computational Project, Number 4 (1994). *Acta Cryst.* D**50**, 760–763.

Cowtan, K. D. & Main, P. (1998). *Acta Cryst.* D**54**, 487–493.

Knight, S. D. (2000). *Acta Cryst.* D**56**, 42–47.

Moroz, O. V., Antson, A. A., Murshudov, G. N, Maitland, N. J., Dodson, G. G., Wilson, K. S., Skibshoj, I., Lukanidin, E. M. & Bronstein, I. B. (2001). *Acta Cryst.* D**57**, 20–29.

Navaza, J. (1994). *Acta Cryst.* A**50**, 157–163.

Otwinowski, Z. (1991). *Proceedings of the CCP4 Study Weekend. Isomorphous Scattering and Anomalous Replacement*, edited by W. Wolf, P. R. Evans & A. G. W. Leslie, pp. 80–86. Warrington: Daresbury Laboratory.

Pannu, N. S., Murshudov, G. N., Dodson, E. J. & Read, R. J. (1998). *Acta Cryst.* D**54**, 1285–1294.

Roper, D. I., Huyton, T., Vagin, A. & Dodson, G. G. (2000). *Proc. Natl Acad. Sci. USA*, **97**, 8921–8925.

Sheldrick, G. M. (1998). *Proceedings of the NATO Advanced Study Institute on Direct Methods for Solving Macromolecular Structures.* Dordrecht: Kluwer Academic Publishers.

Vagin, A., Potterton, E. A., Henrick, K. & Murshudov, G. (2003). In preparation.

Yao, J.-X. (1983). *Acta Cryst.* A**39**, 35–37.