CCP4 in BIOXHIT: Managing and Visualising Project Data

Peter Briggs and Wanjuan Yang

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

Email: p.j.briggs@dl.ac.uk or w.yang@dl.ac.uk

1 Introduction

BIOXHIT is a 4-year integrated project funded within the 6th Framework Programme of the European Commission. The project started in January 2004 and involves over twenty partner institutions, including all the European synchrotrons as well as leading software developers, with the aim of providing an integrated technology platform for Structural Genomics efforts.

One component of this platform is the development of automated structure determination software pipelines that cover the post-data collection stages of structure solution by protein X-ray crystallography. Similar pipelines are also being developed by groups outside of the BIOXHIT project, for example there are a number of CCP4-related efforts including MrBUMP, HAPPy and XIA2 (amongst others). Regardless of their provenance, all these pipelines need to accurately record and track the data that they produce, both for their own operation and for final deposition of the determined structures (considerations that apply equally to more "manual" procedures for determining structures, for example via CCP4i).

CCP4's contribution to BIOXHIT is principally concerned with developing a system for performing this data management in order to address the needs of software pipelines, by building on the existing data management functionality in CCP4i. This article is intended to give a brief overview of progress so far towards this goal.

2 CCP4 in BIOXHIT: managing and visualising project data

2.1 Background

Users of the CCP4 graphical user interface system CCP4i will already be familiar with the job database that forms a key component of the interface. Users divide their work into projects, consisting of a project directory (a UNIX directory or

a Windows folder, depending on the host platform) that is used to hold data files for the project, plus some additional associated data. Most significant among this data is the job database for the project; this is a list of all the jobs (runs of CCP4i tasks) performed within the project. Each job also has associated data that include the taskname, status, date, title, lists of the input and output files, and the parameter file used to run it.

The job database is very simple; however, coupled with a set of tools that allow querying and manipulation of the data it has proved to be one of the most powerful and useful aspects of CCP4i. The fact that every run of a task is automatically recorded in the job database has also contributed to its success; essentially the data is recorded "for free" by the system and so the cost (in terms of the user's time) of using the system to store data is minimal. Expanding this system seemed to be a good starting point to build a more sophisticated project history tracking system.

With this in mind, some of the key considerations for the new system are:

- Implement a single system for both "manual" and automated structure determination
  A single system means that data tracking is standardised regardless of the source, so the same analysis and visualisation tools can be applied.
- Automatically gather as much information as possible from client programs
  One of the strengths of the existing CCP4i system is that jobs are recorded automatically whenever the user runs a task, so this is a good characteristic to try and emulate.
- Provide an "open architecture" that can accommodate heterogenous software components
  One of the potential barriers to non-CCP4i software using the CCP4i system is that it requires that the external program have some Tcl component. Adopting an open architecture reduces this barrier by removing the dependence on a particular programming language.
- Allow flexibility in how the data is stored
  If the way that the data is stored is hidden from the user and the programs that use it, then it is easier to update the database implementation in future (for example if a more sophisticated system such as mySQL is required). This should not be a consideration for the final user of the system however.

2.2 Building the new system

The current implementation of the database in CCP4i is a good start for the new system but suffers from a number of drawbacks, specifically:

- The database is implemented as a simple flat-file (the "database.def" format) which puts limitations on how easily data items can be related to each other, and also becomes less efficient for large amounts of data.
- The database handling functions are embedded within CCP4i and so are not easily accessible from programs that are external to CCP4i.

To address these issues and try to satisfy some of the requirements, a new system is shown in schematically in figure 1, and consists of three principal components:

- The "project database handler" is a small server program that provides the functionality for interacting with the database (and thus separates it from CCP4i, which then becomes just another client application).
- The project database itself, which is only accessed via the handler.
- The "project data visualiser" which provides tools to view and interact with the data in the database.

These components are descibed in more detail in the following sections: the database handler (also called dbccp4i) is described in section 2.3; the database in section 2.4; and the visualiser (dbviewer) in section 2.5.

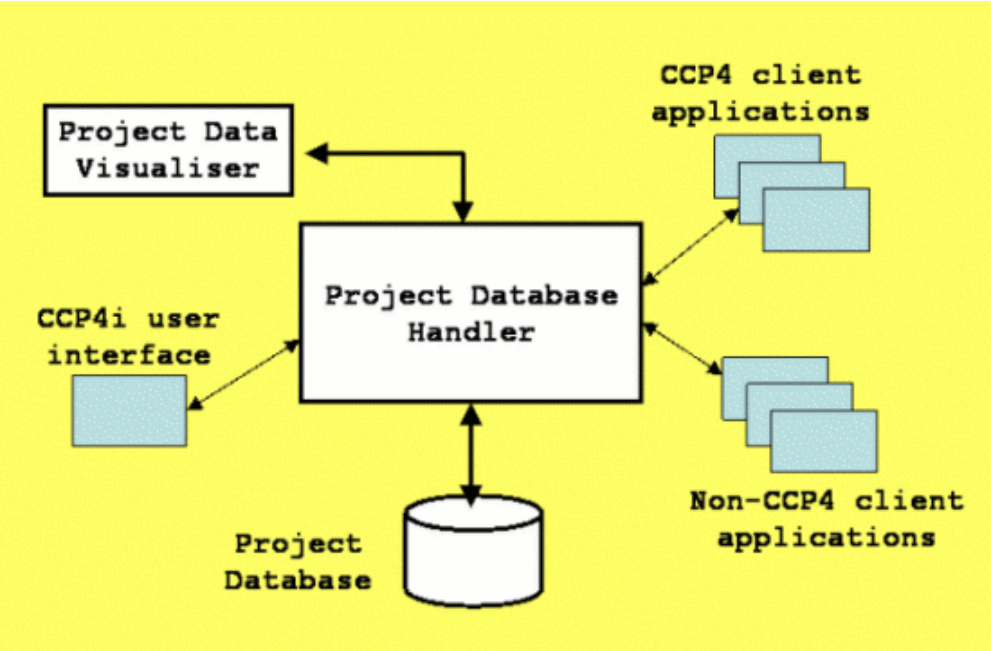

Figure 1: Schematic diagram showing the relationship between the database handler, the database and the visualiser.

2.3 The Project Database Handler: dbccp4i

The Project Database Handler "dbccp4i" is the key component of the system. It is a brokering application that mediates interactions between the project database and external applications: essentially, it is a small server program that should run quietly in the background, acting as a single point of access to the database for external programs like CCP4i (these external programs are referred to generally as client applications).

For most users, the initial version of CCP4i that uses the handler will not appear to be very different from the current version that doesn't; the job database will look the same and the available operations to view, manipulate and delete data will continue to work in the same way.

The real difference will be in making the user's projects and databases available to different programs simultaneously, which is something that is not properly

implemented at present. Initially this will mean that you will be able to have several CCP4is running simultaneously and they will be able to access and manipulate the same project database safely. At the same time the new visualiser application also being developed as part of this project will be able to present different views of the project history data independently of CCP4i (see section 2.5, below).

In the longer term, using the handler will allow programs that are not part of CCP4i or CCP4 to access the database. This would mean that you could perform your structure determination using a mixture of "manual" CCP4i tasks, automated pipeline programs and graphical model building tools, and have all the data history appear in the same project. This should make it easier to monitor a structure determination as it progresses, and help with preparing the data required for deposition at the end.

Also as the database content is expanded in future to include more crystallographic data, it will become easier for programs to share this data between them and thus reduce errors in transferring information through the structure solution process.

Software developers who are interested in more details of using the handler can refer to the "Technical Details" section (section 3) later in this article.

2.4 Database for Project Data & Tracking

The project database handler outlined in the previous section provides the core functionality that will allow multiple applications to access a single database simultaneously. Another facet of this work is to define exactly what form that database takes, and what data it will be able to store.

In the current development the system has focused on "project tracking" (also refered to as "project history") based on the existing CCP4i "projects and jobs" model of the data, which was described briefly in section 2.1. This system has already been seen to provide simple but robust tracking of the steps performed (and the associated data and parameter files) through manual structure solution projects via CCP4i. Reusing the existing "database.def" def-file format for storing the tracking database information in the first instance also ensures backwards compatibility with CCP4i.

Although some modifications to the "database.def" format are required to expand the tracking database to accommodate non-CCP4i programs (for example the name of the program - the "user agent", e.g. an automated pipeline - that writes changes to the database), overall it should provide a sound basis for future developments. For example, combined with the existing data harvesting mechanisms in CCP4 (see for example the CCP4 documentation on harvesting at www.ccp4.ac.uk/dist/html/harvesting.html) it should be possible to make improved tools to facilitate deposition of data to the Protein Databank (some work has been done with the EBI within BIOXHIT to investigate this).

It is important to note that initially, crystallographic data is not being stored directly in the database (only indirectly, for example an MTZ file containing reflection data may be in the project directory and associated with a particular job in the database). However one of the long-term aims of the project is to construct a "rich database" that can combine crystallographic data with the tracking data. To this end some exploratory work (looking at the content of the "rich database" expressed as an SQL schema using an SQLite implementation as an alternative to the def-file format) has been performed, and will form the basis of future developments. See section 3.3 (Development of the "rich database") for more details of these developments.

2.5 Visualisation Tools: dbviewer

Visualisation tools are a general class of software that provide different views of data in order to aid understanding or help to provide new insight. Different visualisers can present the same data in different ways (for example, a PDB file can be viewed in a text editor or in a molecular graphics program) and so can be useful when trying to make sense of large amounts of diverse data.

The starting point for this part of the project is the job display in CCP4i's main window, which can be thought of as a simple visualiser for the job database (figure 2). It shows the project history as a list of jobs in chronological order, with some information about each job, and provides tools to perform additional operations to "drill down" into the data (for example, obtaining a list of input and output files associated with a job).

CCP4i already offers some refinements to this view: for example it is possible to colour job records according to user-specified criteria (figure 3), and the "Search and Sort" tool provides another way again to look at the data (figure 4).

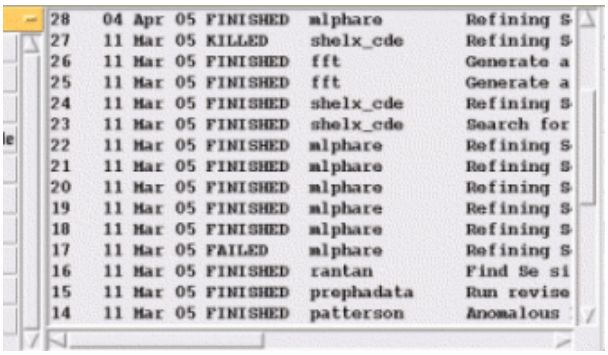

Figure 2 (above): the list of jobs as displayed in the main window of the current version of CCP4i.

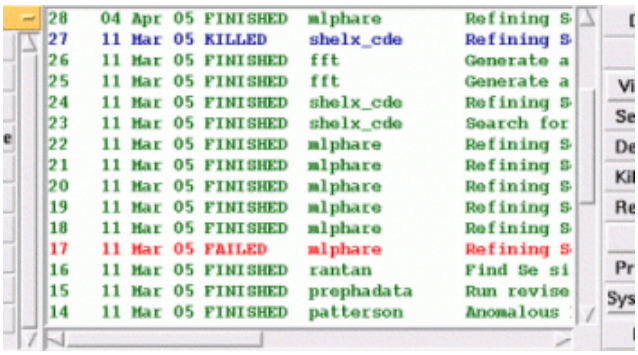

Figure 3 (above): the same list of jobs as in figure 1 with the "custom colours" option turned on, so that jobs with status FINISHED are shown in green; with status FAILED in red; with status KILLED in blue. (Colourising the job listing is an option available under System Administration->Configure Interface.)

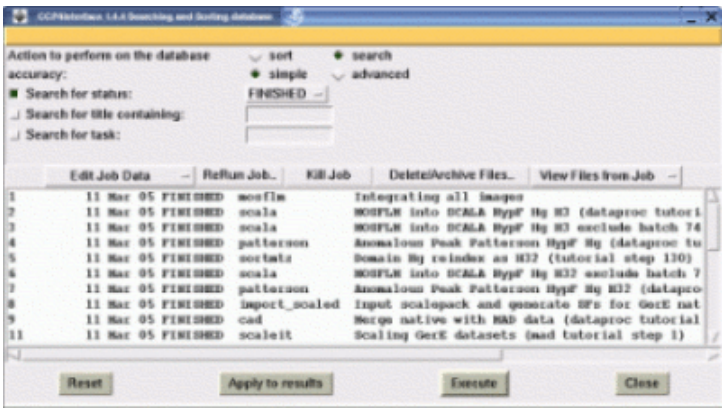

Figure 4 (right): The "Search and Sort" window displaying results of a search for all jobs in the current project with status FINISHED. (The "Search and Sort" tool is available from the menu on the right-hand-side of the main CCP4i window.)

While these tools are very useful, the list-based views also have some limitations, for example as the number of jobs increases it can rapidly become difficult to see how the project has progressed. One of the aims of investigating different visualisations of the project history to help the user better understand their progress and current status.

The "dbviewer" is the first version of a new visualisation tool, and has been developed as a Tcl/Tk client application of the database handler. A screenshot is shown in figure 5, which gives a very different view of the same project shown in the other figures - it presents the project history data as a "directed graph", where 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 various steps.

dbviewer aims to facilitate examination of the project history. Some of its features include:

- Use of different colours to indicate the status of a job (green for completed jobs, red for failed jobs, and yellow for those which are still running)
- The view can be reduced to subsets of jobs via different selection tools, 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.
- "Filters" can be applied to the view, for example to only show jobs that were successfully completed, or which have at least one link to another job.
- Although the visualiser doesn't currently have any functions for modifying the database content, it is able to react to changes in the database in real time. So if a job is added or removed or changes status (for example because CCP4i is running at the same time) then the view will automatically update to reflect this.

Clearly this view and the functions that it offers are useful complements to the existing list-based view in CCP4i.

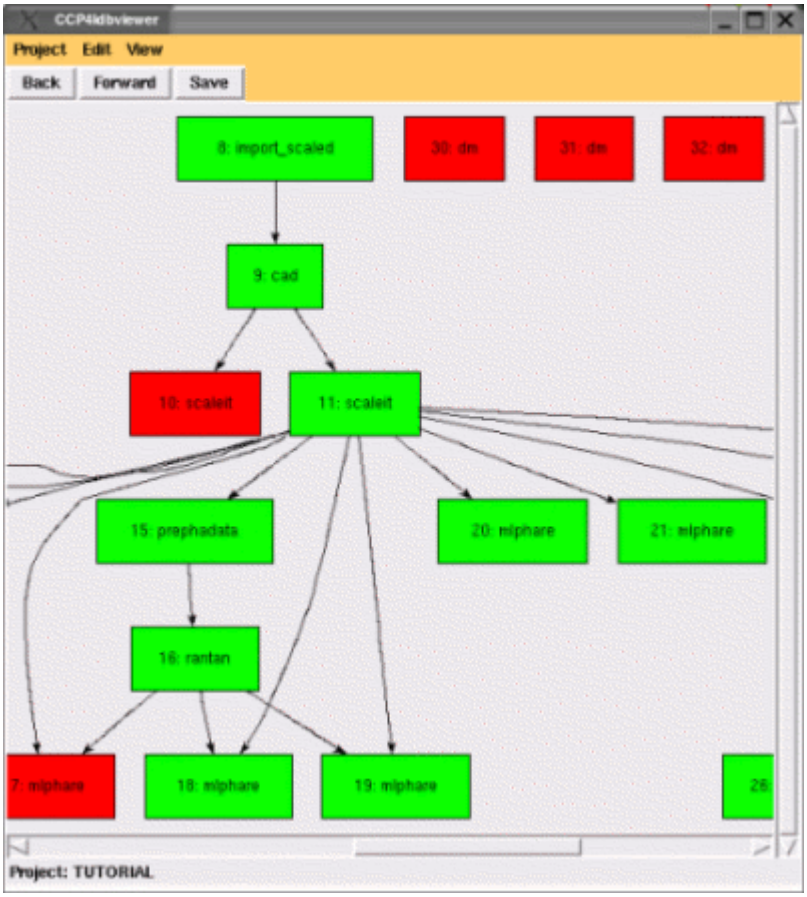

Figure 5: the same project data as shown in figures 2-4 drawn as a directed graph in the dbviewer application. Nodes represent the different jobs with lines between them to indicate the passage of data files.

3 Technical details for software developers

The following sections contain some of the more technical details of the project, which may be of interest to software developers who are considering using some of the components in their own projects. Specifically, they cover how the handler runs (section 3.1), how client applications communicate with the handler to access the database (section 3.2) and some background on the development of the database itself (section 3.3).

For anyone else not interested in these details, this section can safely be skipped.

3.1 How the handler runs

This section aims to clarify the details of how the handler operates in a normal usage environment.

The handler dbccp4i is implemented in Python. Each user runs a single handler process on any machine where they are running client applications that need to access the project database. Only one handler process runs for each user at any one time, and the dbccp4i process automatically manages itself to ensure that this is the case (an example of the singleton design pattern). The client API (see section 3.2) ensures that a client application always establishes connections with the running handler.

Because all client applications run by the same user access the projects through the same handler, and only the handler reads and writes to the project database, there should be no problems with multiple applications accessing the same resource simultaneously for that user.

Note however that there are still issues in the following situations:

- A user accesses the same project from two different machines (for example a project stored in a directory shared between those machines). In this case there will be two handler processes attempting to access the project simultaneously. The handler is designed to "fail safe" in these cases but the operation is not optimal.
- Different users try to access the same project database at the same time, creating a similar situation to the one outlined above.

Further issues occur because the handler operates in CCP4i's database environment, which could (perhaps rather generously) be characterised as a "distributed database architecture". In this environment there is no central database store, instead different data storage components are stored in different locations on the file system:

- Information about the names and locations of each project for a particular user in stored in that user's directories.def file, located in a subdirectory of the user's home area. This file only stores references to projects.
- All the data for a particular project is stored within the project directory for that project, and is independent of any other project.

This distributed setup is extremely flexible and robust, but also means that there are issues for example when one user is interested in sharing a project with another user.

3.2 How client applications communicate with the handler

Communication between the client applications and the handler process are via a simple language expressed in pseudo-XML markup. Requests to the handler from the clients, and responses to those requests, are transmitted using sockets.

One of the functions of the pseudo-XML is to make the communications independent of the programming languages used to implement the handler and the clients. However in practice, dealing directly with the details of socket communications and the pseudo-XML is a potentially significant burden to place on the developer of a client program, and could represent a large barrier to using the system.

To mitigate this, "client API" libraries have been developed that hide the details of the socket and communication protocols from the client. At present client APIs are being developed in Tcl and Python; examples 1 and 2 below show code fragments that illustrate using the APIs within a client application:

```
# Start the handler
if { ![DbStartHandler] } {
    puts "Unable to start the handler"
    exit 1
}
# Connect to the handler
if { ![DbHandlerConnect] } {
    puts "Unable to connect to the
handler"
    exit 1
}
# List the available projects
puts "Projects: [ListProjects]"
...
```

Example 1: Tcl script fragment to illustrate using the client API to connect to the handler and print a list of the user's projects.

```
# Start and connect to the handler
try:
    dbClientAPI.DbStartHandler()
    conn =
dbClientAPI.handlerconnection()
    conn.DbRegister(user,'dummy',True)
except exceptions.Exception,e:
    print "Connection failed with
exception:"
    print str(e)
    sys.exit(1)
# List the available projects
print "Projects:
"+str(conn.ListProjects())
...
```

Example 2: Python script fragment illustrating the same operations as example 1.

The client APIs are intended to perform additional translations to render (for example) lists of data received from the handler, into the appropriate form for the programming language in question, and remove the burden of understanding and implementing the communication protocols from the application developer.

3.3 Developement of the "rich database"

As described previously in section 2.4, the initial version of the tracking database uses the existing CCP4i job database schema along with the def-file based storage. This provides a simple project history/tracking database system with a relatively straightforward practical implementation, which offers a good baseline for futher development.

However the ultimate aim is to provide a system that can accommodate not only the existing tracking data but also combine this with crystallographic data and other data, to produce a so-called "rich database". From discussions with pipeline developers such a database was envisaged as comprising three conceptual components:

- Knowledge Base: consists of the common crystallographic data items used in the software pipeline that are shared between different applications.
- Operational Database: contains arbitrary application-specific data and representations. Examples of these include CCP4i parameter files or serialised Python objects used in an automated pipeline. The essential difference between this and the knowledge base is that data in the operational database are not intended to be shared between applications.
- Tracking Database: stores the history of the data generation in the knowledge and operational databases. Tracking information should include

"data flow" (data items passed between application runs) and "logical flow" (for example, the sequence of application runs that are linked by some kind of logic).

Significant effort was initially expended attempting to create an SQL schema which would describe the content of the knowledge database. However it became clear that making a schema that was generally applicable to even a small number of pipelines (in our case only two!) was a far more difficult undertaking than originally expected. Progress was further hampered in a practical way by not being able to give the developers a "live" implementation of the system to experiment with.

Given this experience, our attention in this area has since focused on implementing a much smaller schema in order to get the technology working first. We then plan to start with a much smaller "seed" schema that can be grown by adding new data items. We are still optimistic that a useful knowledge database can be created, however this will need real usage cases in order to be successful. In this way we hope to avoid the problems that impeded our initial effort in this area.

Efforts to identify and implement an alternative database backend to the CCP4i def-file format have been more fruitful. mySQL was initially considered for this purpose, and although it has many attractive features it was ultimately rejected at this stage for a number of reasons:

- mySQL potentially places a significant burden on the end user in terms of installation, maintenance and usage. This is a particular consideration for CCP4 users who are not typically expert in these areas, and who are used to a very straightforward installation and usage pattern for CCP4, requiring very little setup.
- The centralised nature of mySQL would be difficult to integrate with the distributed database used in CCP4i (see section 3.2).

Instead an embedded SQLite database has been implemented in the handler as a testbed. This allows us to use SQL to express the database schema, while avoiding many of the drawbacks outlined above, and so far the back-end has been road-tested using using a simple SQL version of the current CCP4i database. However in future we to use this to implement the knowledge database.

4 Current Status and Availability

We are currently preparing to make an initial public release version of the handler, the client API libraries for Tcl and Python, and the visualiser. These will all be compatible with the current CCP4i database, and can be run stand-alone.

Once this version is ready it will be announced via the appropriate email lists (for example the CCP4 developers bulletin board, see http://www.ccp4.ac.uk/dev/dev-bb.php). Alternatively if you would like us to

contact you directly when the public release is available then please get in touch with us by email. We are particularly interested in:

- Users of CCP4i who would like to try out the visualiser and provide feedback, and
- Software developers who are interested in using the handler and client APIs to communicate with the tracking database.

Finally, the handler, database and visualiser are among a number of publically-available deliverables being developed for the BIOXHIT project by CCP5. As these CCP4 deliverables are completed, they are being made available via the following URL:

- http://www.ccp4.ac.uk/projects/bioxhit_public/

and the initial version of the three components will be added to this page when they are released.

5 Future Plans & Developments

We are currently working on integrating dbCCP4i into CCP4i, to replace the current CCP4i database handling code with calls to the Tcl client API. A first version of this updated CCP4i should be ready in a few weeks, at which point a developmental version will be made publically available for people to try.

Unfortunately it is not clear whether this will be sufficiently tested in order to be considered ready for release with the next version of the CCP4 suite; however it is possible that a version of dbccp4i and the visualiser dbviewer may be included in CCP4 6.1, and subsequent releases of the suite should include CCP4i working with dbccp4i.

The current version of dbccp4i is missing some functionality that we know users would like to see, including functions to:

- Export and import projects
- Merge and split projects
- Synchronise two copies of a project that have branched
- Move jobs between projects

Aside from this, other longer-term plans include:

- Extending the def file tracking database in order to generalise the data items that are stored; this should make the database less CCP4i-centric,
- Work with external developers to enable their programs to interact with the database via the client API libraries,
- Revisit the knowledge base component of the database, by developing a small "seed" version implemented in SQLite that can be grown more easily with input from collaborators,

- Continue to develop the functionality of the visualiser based on user feedback.

We would also welcome any questions or feedback on the plans or on the work so far.

6 Useful Links

- The BIOXHIT Project website is at www.bioxhit.org
- The CCP4 deliverables from the project are available from http://www.ccp4.ac.uk/projects/bioxhit_public/
- The directed graphs in the visualiser are rendered using a custom Tcl library built on top of the Graphviz "dot" program; see www.graphviz.org.
- The website for the SQLite project can be found at www.sqlite.org. The Python API for SQLite can be obtained via http://initd.org/tracker/pysqlite. The MySQL website is at http://www.mysql.com.

7 Acknowledgements

PJB and WY's work are funded 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. Additional funds are provided from CCLRC Daresbury Laboratory via the CCP4 project.

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

Peter Briggs & Wanjuan Yang, January 15th 2007