LHSG-CT-2003-503420

BioXHIT

A European integrated project to provide a highly effective technology platform for Structural Genomics.

Life Sciences, Genomics and Biotechnology for Health

WP5: Ms5.2.5 Identify items for incorporation into knowledge base database schema
De5.2.8 Implementation of knowledge base schema

Due date of deliverable:	**31.06.2007**
Actual submission date:	**20.07.2007**

Start date of project: **1.1.2004** **Duration: 60 months**

Organisation name of lead contractor for this deliverable: CCLRC-CCP4
Daresbury Laboratory, Warrington WA4 4AD UK **Author** Peter Briggs

Implementation of the Knowledge Base Schema (BIOXHIT milestone 5.2.5 and deliverable 5.2.8)

Peter Briggs, CCP4

1 Introduction

This report describes the demonstration implementation of the dbCCP4i “knowledge base” database.

The knowledge base is intended to contain crystallographic data that is general to a project and that is not application specific. The knowledge database will be modelled using SQL and implemented in an SQLite database within the dbCCP4i system. Our previous attempt at producing an initial database model for crystallographic data was not successful but taught us some key lessons about how to proceed with a second version:

1. Initially try to only model a relatively small amount of data
2. Ensure that you have a working implementation

The demonstration application allows to demonstrate the application of the SQLite technology and to establish mechanisms for reading, writing and updating the knowledge base data. It should also demonstrate the use of the knowledge base infrastructure in a real (but non-critical) application.

The first section deals with identifying crystallographic data items that should be incorporated into the demonstration schema (milestone 5.2.5). The second section describes the details of the demonstration database schema and the implementation of the schema in the database handler dbCCP4i.

2 Milestone 5.2.5: Identification of the Demonstration Knowledge Base Content

The crystallographic data used in the CCP4i "MLPHARE" task was selected as the test case for a demonstration knowledge base implementation. This section describes the data that was identified from this task to be stored in the demonstration database.

2.1 Background to the CCP4i MLPHARE task

The MLPHARE task in CCP4i is used to refine the attributes of heavy atoms in protein derivatives, and/or anomalous scatterers. Within the task there are a number of data items that need to be set by the user before it can be run, including:

- Name of an MTZ file holding the reflection data to refine against
- Whether the data includes anomalous scattering information
- Definitions of each dataset (corresponding to heavy atom derivatives and/or wavelengths)

Each dataset has a number of attributes:

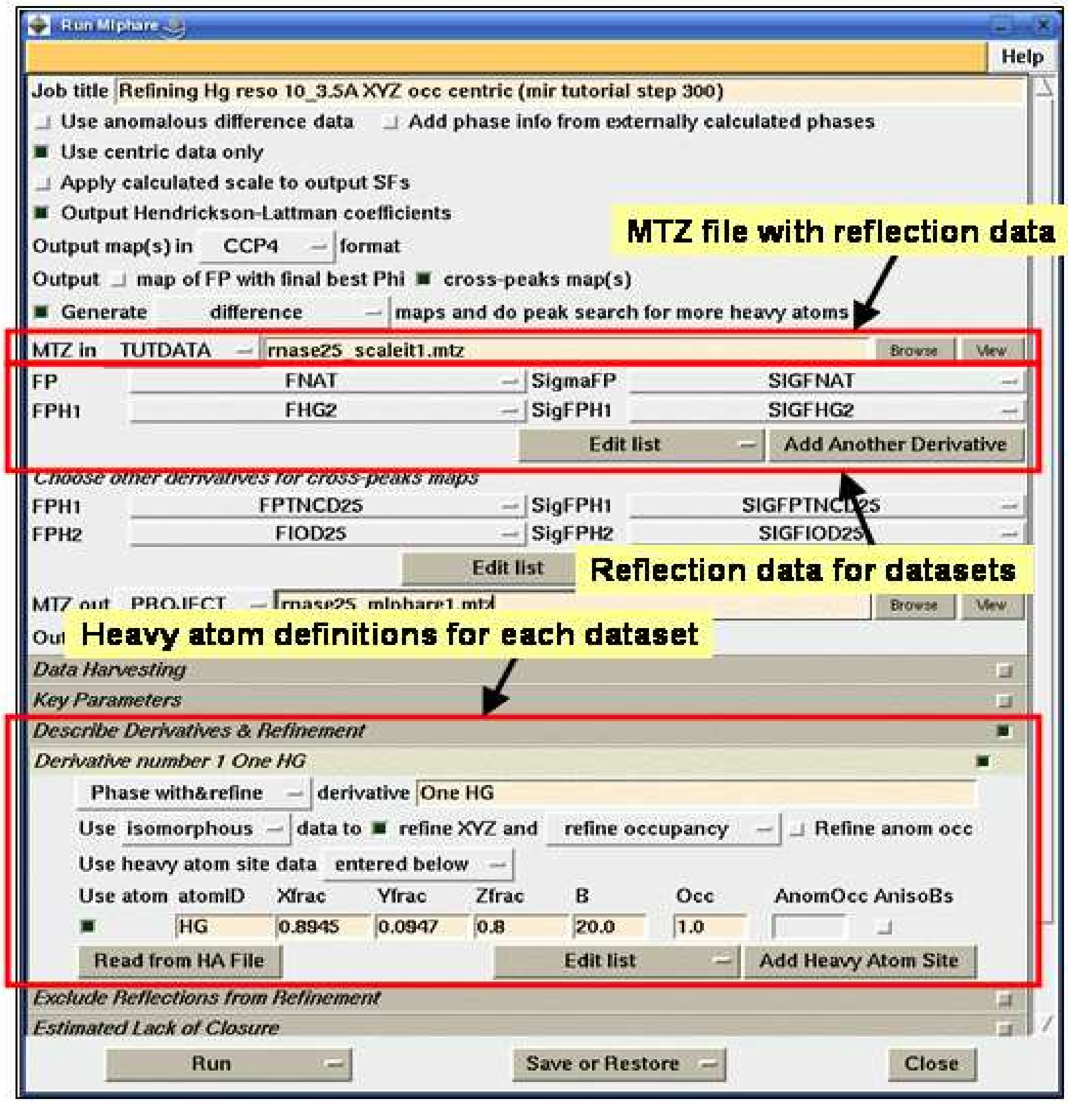
- A user-specified name or title
- A set of associated column labels from the MTZ file to indicate which column contains the data for mean structure amplitudes and anomalous differences
- A set of one or more associated heavy atom definitions

MLPHARE's algorithm assumes that one of the datasets corresponds to the "native" protein, which is the normal situation when performing a multiple isomorphous replacement (MIR) experiment. MLPHARE can also be used to refine heavy atom positions for multiwavelength anomalous diffraction (MAD) experiments, in which case a "pseudo-MIR" approach is used in which one of the datasets must be assigned as the "native" (even though this is meaningless for MAD data).

The heavy atoms can be defined en masse in a single file in CCP4i's ".ha" format, or can be explicitly defined within the interface by the user. The associated data items are:

- The atom type (which must be a valid atom type as defined in the CCP4 scattering library)
- Fractional three-dimensional coordinates x, y, z
- A B-factor
- Occupancy
- Anomalous occupancy
- Optionally, anisotropic B values (set of 6 values)
- Whether the atom is considered to be "in use"

An example of the MLPHARE task interface window is shown in figure 1, with the key data entry areas highlighted in red and captioned.



Job title Refining Hg reso 10_3.5A XYZ occ centric (mir tutorial step 300)
Use anomalous difference data | Add phase info from externally calculated phases
Use centric data only
Apply calculated scale to output SFs
Output Hendrickson-Lattman coefficients
Output map(s) in CCP4 format
Output map of FP with final best Phi | cross-peaks map(s)
Generate difference maps and do peak search for more heavy atoms
MTZ in TUTDATA rnase25_scaleit1.mtz

FP	FNAT	SigmaFP	SIGFNAT
FPH1	FHG2	SigFPH1	SIGFHG2

Choose other derivatives for cross-peaks maps

FPH1	FPTNCD25	SigFPH1	SIGFPTNCD25
FPH2	FIOD25	SigFPH2	SIGFIOD25

MTZ out PROJECT rnase25_mlphare1.mtz
Data Harvesting
Key Parameters
Describe Derivatives & Refinement
Derivative number 1 One HG
Phase with&refine derivative One HG
Use isomorphous data to refine XYZ and refine occupancy | Refine anom occ
Use heavy atom site data entered below

Use atom	atomID	Xfrac	Yfrac	Zfrac	B	Occ	AnomOcc	AnisoBs
■	HG	0.8945	0.0947	0.8	20.0	1.0		

Exclude Reflections from Refinement
Estimated Lack of Closure

Figure 1: Example of the MLPHARE task interface with the relevant data entry points highlighted

2.2 *Outline of application usage of the demonstration knowledge base*

The demonstration application will allow users to enter and edit the general information about the datasets from their experiment into the knowledge base for the current project. When the MLPHARE interface is started there will be an option to populate the entry fields directly from the knowledge base. There will also need to be a way of updating the knowledge base from the output of the MLPHARE run.

The practical steps needed to achieve this are:

- We need to analyse the data that we wish to store in order to define them and their relationships
- We need to organise the data items into a formal knowledge base SQL schema
- We need to implement the SQL schema in the dbCCP4i and provide a Tcl API to allow interactions with the data
- We need to provide a CCP4i interface to enter, review and edit the data in the knowledge base
- We need to provide a mechanism within the MLPHARE interface to populate the fields directly from the knowledge base, with the user only having to select the names of the datasets that have already been defined
- We need to provide a mechanism to allow update of a subset of the knowledge base data from a run of MLPHARE

These steps will inform the design and implementation of the appropriate commands for accessing the data from within the database system.

2.3 Data to store in the demonstration Knowledge Base

The following tables outline the essential data that we would need to store in order to provide the input for the MLPHARE task interface. There are two tables: the dataset table, which holds the "fixed" data derived from the diffraction experiment, and the heavy atom substructure table, which holds references to multiple heavy atom substructures that are generated and refined over multiple runs of MLPHARE.

2.3.1 Dataset Table

Each line in the dataset table describes the reflection data and related parameters derived from a diffraction experiment, and should contain the following data items:

Data item	**Description**	**Details**
Dataset id	A unique identifier within the table.	Arbitrary unique id (used internally in the database)
Dataset name	An identifier for the dataset corresponding to the "derivative name" in the MLPHARE interface.	String of text One per dataset Compulsory Dataset names must be unique within the project, since these will be used by the user to identify and distinguish different datasets (for example when choosing a dataset from a menu) MTZ names can be up to 64 characters long
MTZ file project	The name of a project alias indicating where the source MTZ file that holds the reflection data for the dataset can be found.	String of text One per dataset Compulsory

	Can be blank or "FULL_PATH", in which case the "MTZ filename" must be a full path.	
MTZ filename	The name of the source MTZ file that holds the reflection data for the dataset. This name is combined with the "MTZ file project" to generate the full path name – so if the project alias is blank or "FULL_PATH" then this must be a full path.	Filename/path One per dataset Compulsory Full filenames can sometimes be very long, so this should be able to take at least 200 characters.
Fmean	MTZ column label Indicates the column that holds the mean structure factors	Labels are strings of text One of each label per dataset. Fmean and Sig(Fmean) are compulsory. Dano and Sig(Dano) are optional, but needed for anomalous data. In MTZ files column labels can be up to 30 characters
Sig(Fmean)	MTZ column label Indicates the column with sigmas corresponding to the Fmean values	
Dano	MTZ column label with anomalous difference data	
Sig(Dano)	MTZ column label with the sigmas for the anomalous difference data	
MTZ crystal name	The name of the "crystal" that the data belong to in the MTZ file.	Optional. If supplied then these must all be consistent with the data in the MTZ file. For now they can be used in the data harvesting. 64 character limit for each
MTZ dataset name	The name of the "dataset" that the data belong to in the MTZ file.	
Current HA substructure	The identifier of an entry in the heavy atom substructure data table (see below) indicating the current HA substructure associated with this dataset.	Either blank, or a "HA id" from the heavy atom substructure data table, below.

2.3.2 Heavy atom substructure data table

For the context that MLPHARE is used in, the heavy atom substructure data are not fixed inputs but are refined over several runs of the program. As MLPHARE is run and the data is refined, the heavy atom data will also need to be updated.

To simplify this for the initial version, the data for each heavy atom substructure will be stored in MLPHARE's HA formatted files. The demonstration knowledge base should be able to store any number of heavy atom substructures, and any number should be able to be associated with a particular dataset. As the refinement proceeds, new heavy atom substructures can be added to the table along with the job number (if relevant) which generated the data.

Each line of the heavy atom substructure data table should store the following data items:

Data item	Description	Details
HA id	A unique identifier within the table.	Arbitrary unique id (used internally in the database)
HA file project	The name of a project alias indicating where the file that holds heavy atom substructure data. Can be blank or "FULL_PATH", in which case the "HA filename" must be a full path.	String of text Compulsory.
HA filename	Name of the file containing the data for the current heavy atom substructure, in CCP4i's .ha format. This name is combined with the "HA file project" to generate the full path name – so if the project alias is blank or "FULL_PATH" then this must be a full path.	Filename/path Compulsory. Full file names can be sometimes be quite long, so this should be able to take at least 200 characters.
Job number	The number of a job in the tracking database from which this file was generated.	Optional. Job numbers can be integers or "x.y", where y indicates a subjob of job x.
Dataset id	Id of the dataset that this HA substructure is associated with.	The dataset id will correspond to one of the identifiers defined in the preceding table. Note that more than one heavy atom substructure can be assigned with the same dataset id – this does not have to be unique.

2.3.3 Possible Extensions to the Demonstration Knowledge Base

The following additional data could be stored for each dataset and might be useful in tasks other than MLPHARE:

Data item	Description	Details
F(+)	MTZ column label for structure factors associated with hkl reflections	Labels are strings of text One of each label per dataset. F(+) and F(-) data represent the same information as Fmean and Dano. So Fmean etc could be derived automatically from this data.
Sig(F(+))	MTZ column label with sigmas for F(+) values	
F(-)	MTZ column label for structure factors associated with Friedel mates –h-k-l	
Sig(F(-))	MTZ column label with sigmas for F(-) values	
Crystal cell	Set of cell parameters a, b, c, α, β, γ Cell lengths in Angstroms, cell angles in degrees. One set of parameters per dataset.	All these are optional. If supplied then these must all be consistent with the data in the MTZ file.
Spacegroup	Name of the spacegroup.	

	One spacegroup name per dataset.	
Wavelength	The wavelength of radiation that the diffraction experiment was conducted at. Units are Angstroms	
Fprime	Anomalous scattering factor f' for the heavy atom type in this dataset at the given wavelength	
Fdoubleprime	Anomalous scattering factor f'' for the heavy atom type in this dataset at the given wavelength	

Also, rather than store the heavy atom substructure as a "blob" of data in a file, it might be useful at a later stage to be able to store the individual data items associated with each putative heavy atom. A heavy atom substructure could then be defined as an arbitrary collection of heavy atom records draw from this list.

Although desirable, these extensions would also add an extra level of complexity to the implementation of the demonstration knowledge base that would be unhelpful at this initial stage. They should therefore be relegated to a later stage of the project.

3 Deliverable 5.2.8: Implementation of the Demonstration Knowledge Base

3.1 Introduction

The implementation of the knowledge base consists of a number of parts:

- An SQL schema describing the knowledge base content
- Functions in the database handler to interact with the SQL schema which are exposed via the handler API
- Functions in the client APIs that allow the functions in the handler to be accessed from application programs.

Each of these is described in the following sections.

3.2 Implementation of the demonstration SQL schema

Based on the data items identified for the MLPHARE task the following schema has been implemented:

```
CREATE TABLE IF NOT EXISTS Dataset (
            Dataset_Id INTEGER primary key,
            DatasetName VARCHAR(64) unique not null,
            MTZfileProject VARCHAR(64) not null,
            MTZfileName VARCHAR(200) not null,
            Fmean VARCHAR(30) not null,
            SigFmean VARCHAR(30) not null,
            Dano VARCHAR(30),
            SigDano VARCHAR(30),
            MTZCrystalName VARCHAR(64),
            MTZDatasetName VARCHAR(64),
            CurrentHA INTEGER);

CREATE TABLE IF NOT EXISTS HA (
            HA_Id INTEGER primary key,
            HAfileProject VARCHAR(64) not null,
            HAfileName VARCHAR(200) not null,
            JobNumber VARCHAR(10),
            DatasetId INTEGER);
```

These two tables map onto the data structures described in the earlier section.

3.3 Implementation of the Handler API Functions for the Schema

A set of commands within the database handler need to be implemented in order to allow the data in the schema to be accessed and manipulated.

Two ways of implementing the functions were considered:

- Specific functions for accessing each of the tables and data items explicitly, or
- Generic functions for accessing arbitrary tables and data item access

Specific functions would make for a very transparent handler API, but would require more work if the schema changed (as there would be more functions to update). Generic functions would not be so transparent but would allow for more flexibility in modifying the underlying schema.

Since the schema is a demonstration implementation it was decided that implementing the generic functions was the best choice. The functions are:

Function	Description	Usage
NewTableRecord	Creates and populates new record (=row) in a table for the specified project. Returns the id (=table primary key) for the new record.	NewTableRecord <project> <tablename> <item> <value> [<item> <ivalue>…]
GetAllTableRecords	Return the values stored in all records in a specified table, for a set of named data items.	GetAllTableRecords <project> <tablename> <list-of-data-items>
GetTablePrimaryKey	Return one or more record ids (=primary keys), based on some SQL condition. Used to acquire the id of a record to be used in other operations.	GetTablePrimaryKey <project> <tablename> <condition>
DeleteTableRecord	Remove a record (=row) from the specified table, based on the id (=primary key) of the record.	DeleteTableRecord <project> <tablename> <id>
DeleteTableRecords	Remove several records from the specified table, based on some SQL condition. Used for deleting records from one table that link to a record in another table.	DeleteTableRecords <project> <tablename> <condition>
GetTableData	Fetch the value of a data item for a specified record in a table.	GetTableData <project> <tablename> <id> <item>
SetTableData	Update the value of a data item for a specified record in a table.	SetTableData <project> <tablename> <id> <item> <newvalue>

These are used to build the client API commands described in the following section.

3.4 Implementation of the demonstration Client API Functions

This section first describes how we expect the data in the knowledge base will be accessed, as this informs the client API that the client applications will need, and then the actual commands and usage model that have been implemented.

3.4.1 Abstract Usage Model

Initially the user will need to populate the knowledge base tables with information about their experimental datasets. Since this is the same data as they would need to enter into the MLPHARE interface, this doesn't necessarily entail more work than before (although it does require the creation of a new dedicated interface for initially entering the data).

Each dataset that is created can also optionally have an initial heavy atom substructure file associated with it, however realistically the initial heavy atom substructure data will come from running a program such as SHELX later on in the process. The data entry interface will therefore need to provide options for updating the state of the knowledge base, and possibly also a mechanism for translating the heavy atom substructure data from some other format into a HA format file.

When the user comes to use the MLPHARE interface they will be given the option to use data that they have previously entered into the knowledge base. The MLPHARE interface will give them the option of selecting which of the stored datasets they wish to use simply by selecting the names of the datasets.

Once the MLPHARE task has run, the outputs should include an updated heavy atom substructure in a .ha format file (one per dataset). A reference to this new file can automatically be added to the heavy atom structure table in the knowledge base along with the job id number and a reference to the appropriate line of the dataset table. The HA id number in the “Current HA substructure” data item in the dataset table should also be updated to point to the new substructure line.

3.4.2 Client API Implementation

The client API functions expose the functions in the handler API (previous section) to the client application that wishes to store or retrieve data from the database.

In the case of the demonstration knowledge base, two sets of commands have been implemented:

- Generic commands which map directly onto the generic table functions in the handler API, and
- Specific commands for interacting with the dataset and heavy atom data in the demonstration knowledge base which are built using the generic API commands.

It was also decided initially to only implement the functions in the Tcl client API, rather than in both the Tcl and Python APIs.

The generic commands are:

Generic API command	Description
NewTableRecord	See handler API equivalent
DeleteTableRecord	See handler API equivalent
DeleteTableRecords	See handler API equivalent
SetTableData	See handler API equivalent
GetTableData	See handler API equivalent
GetAllTableData	See handler API equivalent
GetTableRecords	See handler API equivalent
GetTablePrimaryKey	See handler API equivalent

The specific commands implemented using these are:

Specific API command	Description
DefineDataset	Make a new dataset record with values set for each data item, e.g.: *DefineDatabase <project> <DatasetName> <MTZfile> <Fmean> <SigFmean> -dano <Dano> <SigDano> -mtz <MTZCrystalName> <MTZDatasetName>* Note: since a HA substructure cannot be defined for a

	dataset before the dataset itself is defined, the CurrentHA data item would need to be updated using the UpdateHAForDataset command.
ListDatasets	Return a list of the DatasetNames currently in the knowledge base e.g.: *ListDatasets <project>*
DeleteDataset	Remove a dataset previously defined; would also delete the records for any associated HA substructures in the HA table e.g.: *DeleteDataset <project> <DatasetName>*
GetDatasetId	Return the id for a dataset in the Dataset table, based on matching the dataset name, e.g.: *GetDatasetId <project> <DatasetName>*
GetDatasetAttribute	Return the value of a specified data item in the Dataset table for a particular dataset, or of the current HA substructure associated with the dataset, e.g.: *GetDatasetAttribute <project> <DatasetName> <itemName>* where <itemName> is one of the items defined in the Dataset table, or in the HA table. If it is a HA table attribute then it refers to the value for the HA substructure that is the CurrentHA substructure.
GetHAAttribute	Return the value of a specified data item in the HA table, e.g.: *GetHAAttribute <project> <HA_Id> <itemName>*
NewHASubstructure	Defines a new HA substructure, with values set for each data item, e.g.: *NewHASubstructure <project> <HAfile> <DatasetName> -job <JobNumber>*
ListDatasetHASubstructures	Return a list of the HA substructure ids associated with a particular dataset, e.g.: *ListDatasetHASubstructures <project> <DatasetName>*
UpdateHAForDataset	Update CurrentHA for a specified dataset e.g.: *UpdateCurrentHA <project> <DatasetName> <HA_Id>*

3.4.3 Actual Usage Model

The following describes the actual usage of the API commands described previously in an application setting:

1. DefineDataset would be invoked each time the user added a new dataset to the knowledge base. NewHASubstructure would be invoked followed by UpdateHAForDataset to define an initial set of associated heavy atoms, and to set this as the current HA substructure for the new dataset.

2. ListDatasets would provide data to populate menus for selecting which dataset to use for each “derivative” in the MLPHARE interface. GetDatasetAttribute would then be used to acquire the data for the appropriate dataset in order to populate the fields in the interface.

3. After each run of MLPHARE, NewHASubstructure would be invoked to record the refined substructures output from each job. Possibly

UpdateHAForDataset would also be called, to automatically update the current HA dataset associated with the dataset.

4. Any other mechanism for updating the current heavy atom substructure would need to use ListDatasetHAStructures to provide a list of possible HA files for a particular dataset, plus GetHAAttribute to acquire additional information (e.g. the job number). UpdateHAForDataset would be invoked to perform any necessary update to the current HA substructure associated with it.

Availability

The functionality described above is implemented in version 0.3 of the database handler system, and is available for download from the CCP4 BIOXHIT web area:

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

The functionality will also be included in the version 6.1 release of the CCP4 software suite:

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