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

WP5.2: De 5.2.1 Version 1 of independent project database handler application with well-defined interaction protocols

Due date of deliverable: **31.12.2005**
Actual submission date: **31.12.2005**

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

Organisation name of lead contractor for this deliverable: CCP4/CCLRC Daresbury **Author:** Peter Briggs

WP 5.2: Data management and project tracking in structure solution software. Coordinator Peter Briggs (Partner 10), contributing Partners 1C, 10.

De 5.2.1 *Version 1 of independent project database handler application with well-defined interaction protocols*

See supplementary file "de5.2.1_Bioxhit_db-v1.zip" for working version.

Software pipelines such as those outlined in Section 4 will require a component to manage data flow and track project history. This is important in order to make information from all preceding steps available to software components in the pipeline. This WP is concerned with the implementation of a project-tracking database, which can be used within the context of a structure determination software pipeline in order to:

(i) keep track of progress through the structure solution pipeline

(ii) review progress when the pipeline is running in service mode. The owner of the project may wish to query progress through the pipeline, for example a biologist makes a request to the crystallisation service to see whether protein crystals have yet been obtained

(iii) diagnose problems or failures (particularly important for automated procedures). For example if a refinement fails, manual intervention may be required to trace back whether the appropriate model was used in molecular replacement, or whether there were problems in the data processing

(iv) improve procedures by identifying their strengths and weaknesses

REPORT on independent project database handler application with well-defined interaction protocols

This report outlines the structure and usage of the version 1 independent project database handler application, the source code of which is supplied in the supplementary file identified above.

The specification document for the database handler is reported in a separate deliverable D 5.2.3.

Description and Usage of Version 1 of the Project Database Handler

1. Introduction

This document describes the current project tracking system's functionalities, structure and its usage. A copy of the working code (tagged as "version-1_de5_2_1" in the code CVS repository) can be found in the supplementary file *de5.2.1_Bioxhit_db-v1.zip*.

2. Overview

The project tracking system is intended to provide a facility for managing and tracking data during the process of macromolecular structure solution using the technique of X-ray crystallography. Version 1 of the system consists of four parts:

- ClientAPI
- Database (DB) handler
- Database (DB) API, and
- The project database.

Their interactions are represented schematically in figure 1 below.

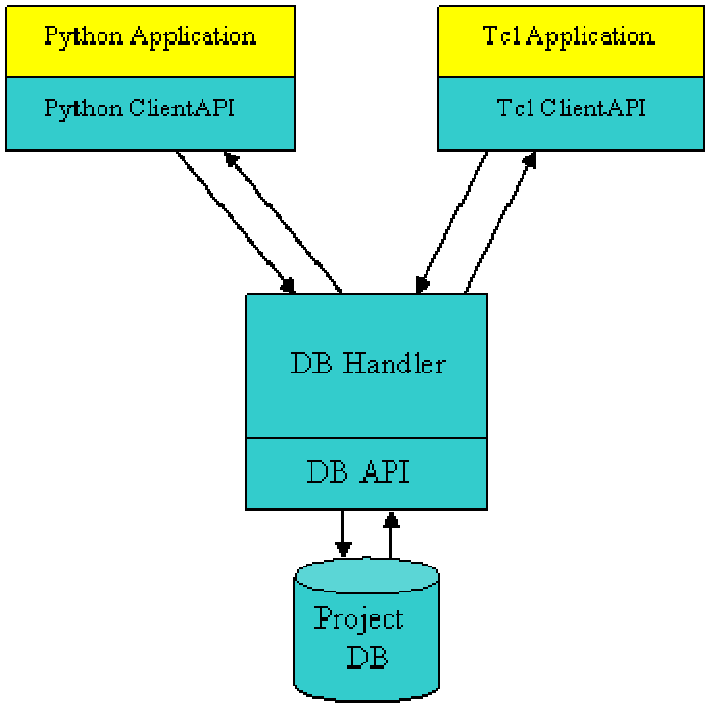

Figure 1: Schematic of the components of the system and their relationships

The DB handler is a server program that handles requests to access the project database from different applications. These applications communicate with the DB handler through a set of functions called the *ClientAPI*. ("API" stands for "application programmer interface" and is more generally a set of programming interfaces provided to allow access to a set of functions implemented in a library.) The DB handler interacts with project database through a second set of functions called *DB API*.

Versions of the ClientAPI functions are implemented in different programming languages, to enable communications with the database from applications written in those languages. In this first version, Python and Tcl ClientAPI implementations are available. An overview of the Client API is given in section 3.

One role of the handler is to be able to hide the details of the DB API from client applications. Thus only a brief discussion of the DB API is given in section 5.

3. ClientAPI functionality and usage

The ClientAPI provides methods/commands to applications that allows those applications to communicate with the project database via the DB handler. In ClientAPI, ***requests*** (both command and arguments) passed to the handler are wrapped in a defined mark-up format, which takes the form of Extensible Markup

Language (XML) strings. ClientAPI sends these requests to DB handler and receives ***responses*** from handler. These responses are also wrapped in standardised XML.

In the Python ClientAPI implementation, the responses are unwrapped and parsed the result is returned to the application. In the current Tcl ClientAPI, however the responses passed to the application remain in XML format and the application must perform the unwrapping – this will be changed in the future.

An example of a request might look like:

```
<db_request>
     <command>OpenDB</command>
     <argument>localhost</argument>
     <argument>wy45</argument>
     <argument>mypassword</argument>
     <argument>PROJECT1</argument>
</db_request>
```

An example of a response example might look like:

```
<db_response>
   <status>ok</status>
   <result>30</result>
</db_response>
```

The communication protocols are discussed further in section 4.4 below.

3.1 Python ClientAPI: technical details

The Python ClientAPI implements a class called ***serverconnection***, from which the access methods are provided. There are 17 methods provided which interface to the version 1 database schema (see section 6):

- **OpenDB(host,username,password,dbname**): Open a specific database. Username and password are their mySQL equivalents.
- **CloseDB(**): Close database.
- **IsOpen(**): Check whether database open or not. Return True if it is open or False otherwise.
- **NewProject(projectname**): Create a new project and set the creation date and time automatically. Return project id.
- **GetProject(projectname**): Open an existing project. Return project id.
- **DeleteProject(projectname**): Delete a project.
- **NewJob(projectname,jobname**): create a new job within the specified project with specified name and set the creation date and time automatically. Return job id.
- **SetData(jobid,name,value**): Create an item within the specified job. Return fact id.
- **GetData(jobid,name**): Get the value of specified item name within specified job. If more than one item with the same name and job id, return the most recent one.
- **ImportDingbat(jobid,name,description,type,value**): Create a new dingbat for a specified job. The dingbat will have name, description, type and value attributes. Return dingbat id.

- **ExportDingbat(jobid,name**): Get the value of specified dingbat name within specified job. If more than one dingbat with the same name and job id, return the most recent one.
- **DescribeDingbat(jobid,name**): Return the description and type of specified dingbat for a specified job.
- **DeleteDingbat(jobid,name**): Remove a dingbat and its associated data from a specific project for a specific job and specific name.
- **ListProjects(**): Return a list of all the projects in the database.
- **ListJobs(projectname**): Return a list of jobs in a specified projects.
- **ListItems(jobid**): Return a list of facts associated with a specified job in a specific project.
- **Version(**): Return the version of MySQLdb package.

Python applications that want to communicate with the DB handler and thus access project database should include the following line:

```
import ClientAPI
```

To make a connection to the handler, the first thing to do is create an instance of serverconnection. For example,

```
myconn = ClientAPI.serverconnection()
```

The methods in *serverconnection* then can be used. *OpenDB* should be called first before other storage or retrieval data methods can be used. When all operations are complete, CloseDB() should be called to close the connection.

3.1.1 Example

```
myconn.OpenDB(host,username,password,dbname)
myconn.NewProject(projectname)
myconn.NewJob(projectname,jobname)
….
myconn.Close()
```

3.2 Tcl ClientAPI:

The Tcl ClientAPI provides a set of procedures that Tcl applications can use to connect to the DB handler and thus access the project database.

Currently 13 procedures are available which interface to the version 1 database schema (see section 6):

- **OpenConnection host port:** Open a connection to the handler. Return a client socket.
- **IsOpen sock:** Check whether database open or not. Return True if it is open or False otherwise.
- **CloseConnection sock:** Close the connection to the handler.
- **OpenDB sock host username password dbname:** Open a specific database.
- **NewProject sock projectname:** Create a new project and set the creation date and time automatically. Return project id.

- **NewJob sock projectname jobname:** Create a new job within the specified project with specified name and set the creation date and time automatically. Return job id.
- **SetData sock jobid name value:** Create an item within the specified job. Return fact id.
- **GetData sock jobid name:** Get the value of specified item name within specified job. If more than one item with the same name and job id, return the most recent one.
- **ListProjects sock:** Return a list of all the projects in the database.
- **ListJobs sock projectname:** Return a list of jobs in a specified project.
- **ListItems sock jobid:** Return a list of facts associated with a specified job in a specific project.
- **GetProject sock projectname:** Open an existing project. Return project id.
- **Version sock:** Return the version of the MySQLdb package.

Tcl applications that want to connect DB handler and thus access the project database should include the following line:

```
source clientAPI.tcl
```

To make a connection to DB handler, use the command *OpenConnection*. A client socket will be returned. Using the client socket, the other commands can be accessed. *OpenDB* should be invoked before other storing and retrieving data commands are used. *CloseConnection* should be used to close the connection at the end.

3.2.1 Example

```
set s [ OpenConnection host port ]
OpenDB $s host username password dbname
NewProject $s projectname
… …
CloseConnection $s
```

4. DB handler

The DB handler is a server program that accepts requests in a defined format from external applications that tell it how to manipulate the data stored in the database, and issues responses to those requests.

The following sections give an overview of the workings of the server, how it is started, and some of the associated issues.

4.1 How it works

When the DB handler is running, it listens for requests from clients. When it receives a request, the handler unwraps the request and gets the command and arguments. It then verifies whether the command structure is valid or not (note that if applications communicate with the handler through one of the ClientAPIs described above, the command structure should always be valid). The DB handler then calls the DB API in order to perform the requested database operation, and gets back a return value. The DB handler wraps this result in XML and returns it to the client.

4.2 Persistence of connections

The DB handler can process requests from several different clients simultaneously. While the DB handler is running, each of the client applications can establish a connection to the handler that persists for as many operations over as long a time period as they wish – it is left to the client to decide when to close the connection.

4.3 Authentication

In this first of the DB handler itself, authentication issues are not considered within the handler or database implementation. Any client application can connect to the DB handler provided that the requests that are issued are correct. However, client applications must provide username and password to be able to access MySQL database backend, which means they must have an account in the database. So authentication is controlled through MySQL database.

4.4 Communication protocol

Requests sent from client applications to the handler, and the responses returned from the handler to the applications, are wrapped in a pre-defined XML format that is outlined here.

Requests have the following general wrapping:

```
<db_request>
      <command>…</command>
      <argument>… </argument>
      <argument>… </argument>
      …
</db_request>
```

The root element is <db_request>, there is one <command> element and zero or more <argument> elements (depending on the command). The command itself is one of the methods/commands provided in ClientAPI, and each argument corresponds to one of the arguments of the method/command.

Responses have the following general wrapping:

```
<db_response>
   <status>…</status>
   <result>…</result>
   <result>…</result>
   …
</db_response>
```

The root element is <db_response>, there is one <status> element and one or more <result> elements. Status can be either ‘ok’ or ‘failed’, and reflects the success or otherwise of carrying out the requested operation. The result is the return value from database.

4.5 How to run the DB handler

To run an instance of the DB handler, you must have DB handler itself (provided as server.py) and the DB API (provided in db.py and dbapi.py). The project database must be also have been set up in advance in the mySQL server (this is described in section 6).

To run the handler, type the following command at the prompt:

```
python server.py [ -debug]
```

With the *–debug* option, the diagnostic output messages will displayed on the screen. Without this option the handler runs silently, and output messages are written to the log file *handler_log.txt*.

5. DB API

The DB API implements functions that interact directly with the database backend. Currently a DB API is only provided for a MySQL database backend. It implements a DB class that has a range of methods mapping onto with the methods provided in ClientAPI.

As client applications should communicate with the database via the handler, the details of the DB API do not need to be known to them. You are referred to the source code for more information on how these are implemented.

6. The Project Database

The version 1 project database is implemented in MySQL. Currently, the schema consists of a single MySQL database comprising four tables: ***projects***, ***jobs***, ***facts*** and ***dingbats***.

The relationship between these tables is:

- The database contains one or more projects
- Each project contains one or more jobs
- Each job has one or more facts and one or more dingbats

Table 1 describes the tables and their functions in more detail.

Tables	**Description**	**Attributes**
Projects	A collection of jobs that have something in common, for example steps towards determining the structure of a single protein. Project names must be unique.	• project_id (unique) • project_name (unique) • start_date • end_date
Jobs	A job is an instance of an operation, such as a run of a program or script.	• job_id (unique) • job_name • project_id • start_date

		• end_date
Facts	A fact is some item of information associated with a job, for example the solvent content or cell parameters. Note that facts do not have to have unique names.	• fact_id • job_id • name • value • epoch
Dingbats	A dingbat is storage of some complex object, for example a pickled python object. Note that dingbats do not have to have unique names. Dingbats are stored as base64 encoded strings. ClientAPI encodes the input object from the application before transmission to the handler. ClientAPI decodes the string when retrieving the encoded object from database.	• dingbat_id • job_id • name • value • description • type

To set up the project database, run the script `schema.sql`, which contains the mySQL commands defining the tables.

7. Project Data Explorer

The project data explorer is a prototype visualiser. The explorer provides three windows, one each for listing projects, jobs and facts. The "projects" window lists the names and start dates for all projects in the database. The user can select one of the projects, and the list of associated jobs (names and start dates) within that project will be displayed in the "jobs" window. Selecting one of the jobs causes the list of facts associated with that job to be displayed in the "facts" window. The "refresh" button in each window allows updated information to be displayed. No display of dingbats is implemented.

This first version of the data explorer is implemented in Python using TkInter, and connects to the MySQL database directly. In the future, this will be changed to connect to DB handler and act as a client application. For this version the explorer is provided simply to allow users to easily look at the database content.

8. Authors and acknowledgements

Wanjuan Yang has performed this work with substantial input and guidance from Peter Briggs. The funding for this work has been provided from the BIOXHIT Project funded by the European Commission with its FP6 Programme within the thematic area "Life sciences, genomics and biotechnology for health" contract number LHSG-CT-2003-503420. Graeme Winter and Daniel Rolfe also provided useful discussions and feedback.