Technical Report CESGA-2007-002

Quantum Chemistry common data format Q5COST and OpenBabel: A first answer to interoperability in Quantum Chemistry

Aurelio Rodríguez
CESGA. Centro de Supercomputación de Galicia
e-mail: aurelio@cesga.es
Mario Valle, Ugo Varetto
CSCS. Swiss National Supercomputing Center
Email: mvalle@cscs.ch, uvaretto@cscs.ch

Abtract: The **Q5COST** format has been designed and implemented with the aim of develop a Common Data Format for promoting **code interoperability in Quantum Chemistry**. Open Babel is a chemical toolbox designed to speak the many languages of chemical data. This document try to be a step by step guide about the use of the **Q5COST** binary format (based on **HDF5**, a unique technology suite that makes possible the management of extremely large and complex data collections) and how to integrate it on **OpenBabel**.

Keywords: Quantum Chemistry, Q5COST, OpenBabel,HDF5.

Este trabajo ha sido cofinanciado por el FSE (Fondo Social Europeo)

Identificador documento:	DOC_APL_Q5COST-OpenBabel_V0
Fecha:	30/11/2007
Autores:	Manuel Aurelio Rodríguez López, Ugo Varetto, Mario Valle
Responsable:	Andrés Gómez
Status:	

Historia de Revisiones

Versión	Autor	Descripción
0	Aurelio Rodríguez	Technical Report Development

Index

1 Introduction

A key development made by the DeciQ[i] WG of the COST-D37[ii] action has been the design and implementation of a Common Data Format for promoting code interoperability in Quantum Chemistry.

This data model is based on the Q5COST format (HDF5-based format supporting large binary datasets).

This new data format needs to be implemented in the most common Quantum Chemistry packages and also in several tools used nowadays to prepare the input and to visualize the results. This technical report is focused on this last topic.

Open Babel is a chemical toolbox designed to speak the many languages of chemical data. It is an open, collaborative project that allows for anyone to search, convert, analyze, or store data from molecular modeling, chemistry, solid-state materials, biochemistry, or related areas. Nowadays many Quantum Chemistry visualization and utility tools are based on or use the Open Babel toolbox. So, there is a clear necessity of adding the Q5COST Common Data Format to Open Babel.

Hence, the objective of this technical report is to give a detailed description of the Q5COST format and the process followed to add the new file format Q5COST to Open Babel. With that, software packages like VMD (a molecular visualization program widely used by the biochemistry and bioinformatics community) or MOLEKEL (an open source (GPL) multiplatform molecular visualization program developed by the CSCS and used by the Quantum Chemistry community) could support the common data format.

2 Q5COST format description[iii]

The Q5COST data file structure has been specifically defined with the aim of making code interoperability easier in the scientific context of interest of the quantum chemistry.

Q5COST is a common interchange format for large binary data coming from quantum chemistry calculations. It is flexible and extensible, so we can design it in an incremental way. The data format consists of a collection of chemical objects related within a hierarchical structure in a logical containment relationship as reported in Figure 1. The green boxes are containers (groups), and the yellow boxes are data (data sets), while the metadata (attributes describing the data) are listed in the white labels.

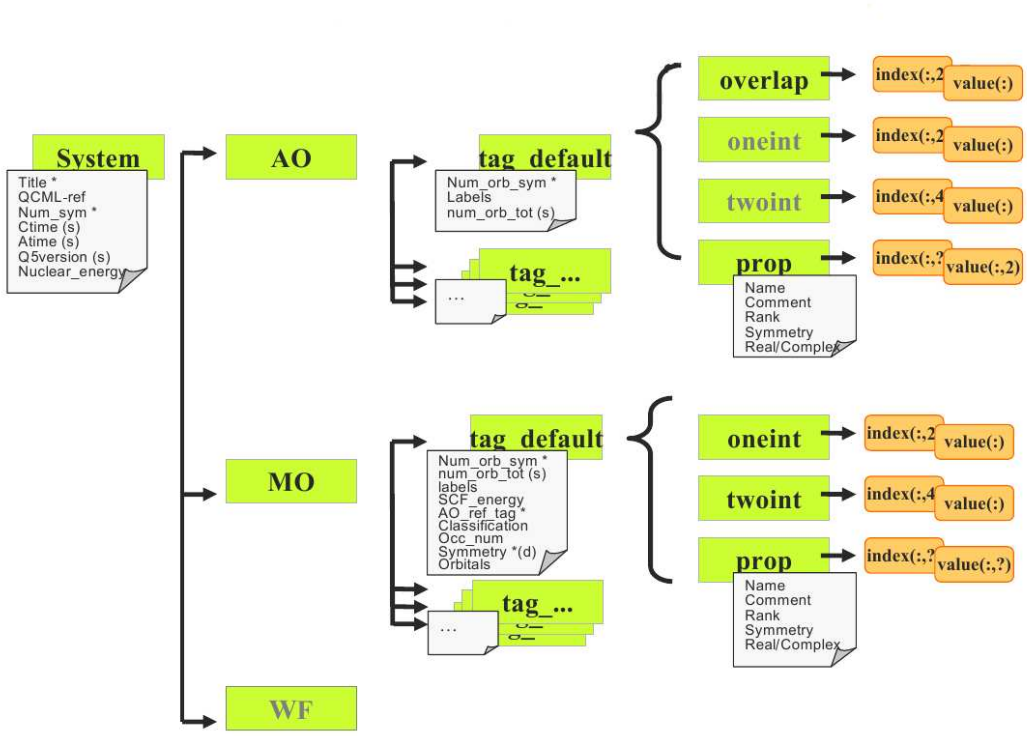

Figure 1Q5COST hierarchical structure

The QC-ml format will be integrated on the general Q5COST format in the followings releases so the possible implementation of the QC-ml format in Open Babel will wait until then.

2.1 *The Data Model*

This extensible data model has been constructed using as a basis the experience of the research groups involved in the Q5COST development. After a preliminary analysis, some clear points have emerged.

The first point is that many different types of simple and small data must be handled (nuclear energy, molecular orbital labels, molecular symmetry, and so on). We will refer to these data as "metadata", in order to distinguish them from the real large information on the chemical system such as the integral values.

Metadata represent well-known chemical entities and belong to three generic data classes: scalars, vectors, and matrices. For example, the nuclear repulsion energy is a floating point scalar, molecular orbitals are an (N,M) floating point matrix, the associated orbital energies are a floating point vector, the molecular orbital labels are a vector of strings, and so on. The library should provide an interface for accessing these data both as generic or specialized entities.

A second point is that in quantum chemistry large matrices with an arbitrary number of indices (rank-n arrays) are very common data structures. These data usually scale aggressively with the system size, and they are normally accessed with a "chunked" approach (i.e., using well-defined blocks of data). This is not only the case of entities like two-electron integrals, or atomic orbital overlap, but also other more application specific information, like the four particle density matrix.

These large array data share common features:

- They are usually integrals, whose evaluation involves one or more operators and a given (large) number of functions. These functions are referenced by the indices of the matrix. For example, two-electron integrals on the molecular orbital basis are stored as a rank-4 array with indices referring to the molecular orbitals; in the case of atomic basis set overlap integrals, the indices refer to the atomic basis set orbitals.
- The rank of the matrix depends on the physical meaning of the integrals. The atomic basis set overlap is described by two indices and can be stored as a rank-2 array, two-electron integrals have four indices imposing a rank-4 array, and the four particle density matrix has eight indices, requiring a rank-8 array.
- Additional information is needed to describe the operator involved (for example its symmetry) and the entities the indices refer to.

Thus, all these data objects can be described by a "generic property" object, provided that we define the matrix rank and the involved operator(s) and basis functions.

Since some of these "properties" are well-known chemical entities, and chemists are used to referring to them by name, the developer have provided a specific library to access to most of them (overlap, one-electron integrals, two-electron integrals...), in addition to an interface to the "generic property" for handling other properties not explicitly provided by the library. This should ensure both ease of use and general adaptability of the library to either alternative or future theoretical developments.

The last point is that all these chemical objects are related within a hierarchical structure, and logical containment relations can be defined for them.

A first (root) container, named System, represents the molecular system as defined by its structural data (chemical composition and spatial geometry). To this container can be associated all the metadata that are invariant at the level.

A system can contain several "Domains". The role of the Domain is to group together Property entities whose indices conceptually refer to the same kind of functions. Three

Domains have been recognized as fundamental: AO for Atomic Orbital, MO for Molecular Orbitals, and WF for Wave Function.

The AO Domain holds properties referring to the atomic basis set functions: overlap, one-electron, and two-electron integrals on the atomic basis set, in addition to the generic property. The invariant metadata consist of information on the atomic orbitals, like their number, the labels, and symmetry.
The MO Domain holds properties referring to molecular orbitals: one-electron and two-electron integrals on the MO basis set, in addition to the generic property. The descriptive metadata for the domain refer to the MO basis description: their number, labels, and symmetry, the AO basis they were derived from, the matrix collecting the coefficients of the MO expansion on the AO basis, orbital energies, classification, and occupation numbers.
The WF Domain holds properties referring to the electronic states. The complete definition of this container is not available yet. It is still the subject of research and development.

For each of the domains, different occurrences can be defined by means of an identifier (tag) chosen by the user, with a default value if no tag is provided. The aim is to provide storage for multiple entries, like in the case of multiple molecular orbitals in the MO Domain or multiple basis sets in the AO Domain.

The bottom level of the hierarchical scheme is made of the properties. Even if from the user's point of view many different "properties" are available, all of them are different instances of the same "generic property" object. This object holds the true data, i.e., the integral values and the corresponding index values. Also here, in order to fully define the nature of the actual property, some metadata are needed: name, rank, symmetry, and type (i.e., real, imaginary, or complex).

3 Q5COST library

The Q5COST library provides read and write access to files defined in accordance with the data model described before (Q5COST data model). It provides a specifically designed high level access for quantum chemistry developers. The rationale is to provide a FORTRAN interface based on well-known chemical entities, rather than groups or data sets like in the original HDF5 interface. HDF5 takes care of the low level management of the file, and Q5COST provides the high-level Application Programmer Interface for storage and retrieval of chemical entities.

3.1 *Library Structure*

The library is written in FORTRAN95 and consists of several modules, each one providing different facilities. The most important modules are as follows:

- **Q5COST**: defines the high-level API. This module provides subroutines designed to be at the disposal of the final programmer.
- **Q5Core**: provides a wrapping facility for HDF5 routines, in order to perform additional useful services like reference counting and debugging. It also provides simplified routines to perform frequently used low-level tasks.

- **Q5Error**: provides facilities for high level debugging of library and client codes. This module implements a ring buffer for error messages, different logging levels, generic reference counting for catching memory leaks, and a subroutine call stack trace.

The names of the subroutines in each module are identified by an appropriate prefix and have been chosen to provide an explicit and intention revealing interface to the entities described in the previous section. Although FORTRAN 95 does not allow object oriented (OO) programming, some OO concepts have been used in the development of the library but taking into account the possible procedural programming background of future developers. The state is preserved in the HDF5 file, and subroutines refer to the file directly through the HDF5 file identifier, an easier concept for FORTRAN programmers more used to file descriptors.

3.2 *The Q5COST Module*

This module is the main reference for the final user. It provides subroutines to read and write HDF5 files in the Q5COST format with a high level of abstraction. Using this library the users can deal with high level concepts without worrying about low level implementation details. If a finer access is required for the underlying HDF5 file, the Q5Core module provides this type of access in a simpler way with respect to the raw HDF5 routines.

All the routines in the Q5COST module have the q5cost_prefix, and they are organized in several classes:

Init: initialize and deinitialize the library within the program.

File: create, open, close the .q5 file, and write/get root attributes, like creation time, access time, and file version.

System: create or check the existence of the System and set/get the specific attributes

AO: create or check the existence of a given occurrence of the AO Domain and set/get its attributes

AOOverlap: create the folder, read and write data for the atomic basis set overlap property

AOOneInt: create the folder, read and write data for the one-electron integrals in atomic aorbitals basis

AOTwoInt: create the folder, read and write the data for the two-electron integrals in atomic orbitals basis

MO: create or check the existence of a given occurrence of the MO Domain and set/get its attributes

MOOneInt: create the folder, read and write data for the one-electron integrals in molecular orbitals basis

MOTwoInt: create the folder, read and write the data for the two-electron integrals in molecular orbitals basis

WF: create or check the existence of the “WF” domain and set/get its attributes

Property: create the folder, read and write data for a generic property. The name, domain, rank, and type have to be defined by the user.

Additional routines are available for the generic access to the "Property" class, allowing the management of user defined properties. Subroutines like AOOverlap, MOOneInt, and MOTwoInt contain calls to these property routines, passing the specific parameters of the involved property.

The routines of the Q5COST module provide a context-based access to chemical entities. This access is converted into a path-based access, creating an appropriate layout for HDF5 groups, data sets, and attributes, and writing the user provided data into the file. Some data are provided automatically by the library, like the creation or access time and the Q5COST library version.

One important aspect of this format is that the user is not forced to enter all the quantities; he can store the quantities that are actually available, or in which he is interested, and add other data later when available. Constraint checks are however mandatory in order to ensure basic file consistency.

For example, a MO Domain can be created only if a System and an AO Domain exist, in order to guarantee the presence of fundamental data, like the number of symmetry species and the number of basis functions for each symmetry species.

3.3 *The Q5Core Module*

The Q5Core module is a low level module designed to provide wrapping facilities between HDF5 and Q5COST. At the moment it is focused on providing additional debug information, reference counting for HDF5 objects, additional low-level API for simplifying common tasks, and so on. This module provides path-based management of scalar, vector, and matrix entities (in contrast with the context-based approach of the Q5COST module, which focuses on chemical concepts rather than HDF5 path). It also provides routines for the easy handling of the Property data (indices and values), relative to a CompactMatrix class (CM).

End users in general should not access Q5Core module routines.

The Q5Core module guarantees the transparency of the Q5COST data model with respect to the underlying technology.

In case we decide to use another storage format in place of HDF5, only this module should be modified. The Q5COST module, i.e., the end user interface, remains unchanged, being independent of the low-level format.

3.4 *The Q5Error Module*

The Q5Error module provides subroutines for debugging and monitoring the behavior of the library and the application code. A ring buffer is provided to keep track of error messages generated by the library. A verbosity level can be set, from totally silent to highly verbose; in the latter case each subroutine call and return is reported in the buffer. Moreover, a stack for backtracking has been implemented to keep track of the call tree. The tree is printed out when an error occurs or when error reporting is requested. Different specific error codes have been provided for, to report anomalous behavior of the application code or of the library itself. The error codes are defined as

numeric parameters and report situations ranging from invalid parameters to nonexistence of some information in the file. The presence of an error condition is returned to the application code through the last parameter of each subroutine.

A full detailed description of the Q5COST library API can be found at the DeciQ site[iv].

4 OpenBabel, functionalities

Among other features, Open Babel offers:

- Ready-to-use programs for converting files, molecular searching, chiral detection, and superimposing molecules:
  - **babel** -- a converter for chemistry and molecular modeling data files
  - **obchiral** -- print molecular chirality information
  - **obfit** -- superimpose two molecules based on a SMARTS pattern
  - **obgrep** -- an advanced molecular grep program using SMARTS
  - **obprop** -- print standard molecular properties
  - **obrotate** -- batch-rotate dihedral angles matching SMARTS patterns
- Complete programmer's toolkit including C++, Perl, Python interfaces for easy custom software development
  - In addition to serving as a set of user-level tools, Open Babel offers a C++ library and interface in other languages e.g., Perl and Python for general chemical software development, both in-house and to encourage open source chemistry packages. As such, it is also a founding member of the Blue Obelisk movement[v], which shares cheminformatics data, algorithms and more.
- Support for a huge variety of common chemical file formats, including SDF/MOL, Sybyl mol2, PDB, SMILES, XYZ, CML... [vi]
- Automatic recognition of file type based on filename extension
- Implementation of Daylight SMARTS molecular matching syntax
- Batch conversion for multiple molecules in one file (e.g., splitting, merging, batch operation)
- Gasteiger-Marsili partial charge calculation
- Hydrogen addition and deletion
- Isotope support, calculation of average and exact masses
- Flexible atom typer and perception of multiple bonds from atomic coordinates
- Automatic feature perception (rings, bonds, hybridization, aromaticity)
- Open-source/Free Software under the GNU General Public License

- Cross platform (Windows, Linux, Mac OS X, SGI, Solaris, PlayStation...)

4.1 *Open Babel API*

Open Babel is a full chemical software toolbox. In addition to converting file formats, it offers a complete programming library for developing chemistry software. The library is written primarily in C++ and also offers interfaces to other languages (e.g., Perl and Python) using essentially the same API.

The heart of Open Babel lies in the OBMol, OBAtom, and OBBond classes, which handle operations on atoms, bonds and molecules. Newcomers should start with looking at the OBMol class, designed to store the basic information in a molecule and to perceive information about a molecule.

One of the key philosophies in the code is that transformations and automatic perception of properties are performed in a "lazy" manner. That is, until you call for partial atomic charges, no charges are calculated. This ensures faster transformations of chemical data -- properties that are not needed for your code will typically not be calculated. When such data is needed, appropriate routines are called, and a "flag" is set (e.g., via OBMol::SetFlag or OBAtom::SetFlag etc.) so that the code is only run once.

Arbitrary custom data and text descriptors can be stored in any atom, bond, molecule, or residue using the OBGenericData or OBPairData classes. That it is particularly important for this report aim, *through a OBGenericData subclass it is possible to give support to all the data accessible by the Q5COST library* .

Conversion between various chemical file formats is accomplished through the OBConversion and OBFormat classes, often through use of the OBMoleculeFormat subclass which is designed for easy read/write access to one or more OBMol objects. The philosophy of the file format codes is to parse as much chemical information from a given file as possible (no data left behind) and ideally any perception or transformations will occur when writing to some other format later.

Detailed documentation of all main Open Babel classes can be found at the Open Babel web site[vii].

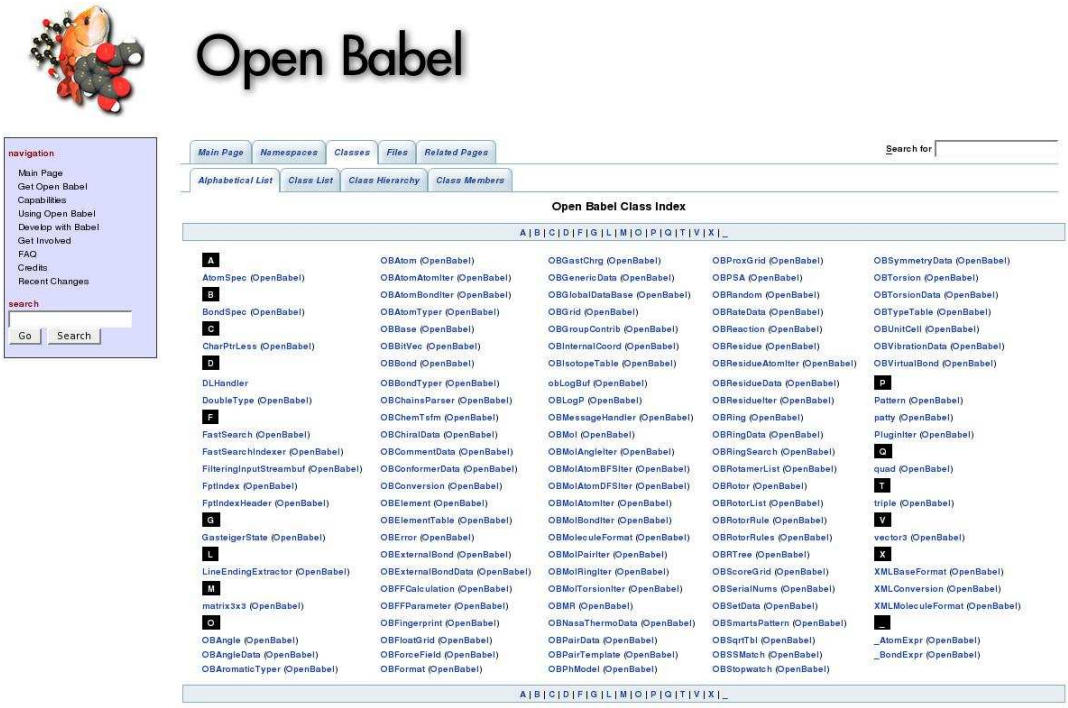

Figure 2 Open Babel API

5 The Q5COST Reader implementation in OpenBabel

This is the report's main subject. We will try to emphasize the most important points of the process and at the same time, to collect the set of "Tips & Tricks" needed during the process.

5.1 *Preliminary software installation:*

It is required a full Q5COST library installation. Detailed installation instructions can be found at the DeciQ site[viii]. The Intel Fortran compiler have been used along all the procedure.

Additionally to these instructions it is recommended to use the following script to call the configure tool:

```
#!/bin/bash
#default compiler ifort
mkdir include lib
export FC="ifort -fPIC"
export LIBGCC=PATH_TO_libgcc.a
export LIBHDF5= PATH_TO_libhdf5.a
export LIBHDF5_F= PATH_TO_libhdf5_fortran.a
./configure
```

Otherwise the configure tool will use a "find" command to set these variables. Also all the remaining steps needs the environment variable Q5COSTPATH set to the Q5COST library installation path.

TIP: A shared Q5COST library will be needed for the Open Babel integration. For the sake of simplicity this dynamical library should content (no dependency on) the HDF5 library. A command like this one should be run from the $Q5COSTPATH/lib folder:

```
> ifort -shared -o libq5cost.so ../*.o  $HDF5DIR/lib/libhdf5_fortran.a $HDF5DIR/lib/libhdf5.a -lz
```

Finally uncompress the Open Babel source code distribution. It can be downloaded from the Open Babel site[ix].

5.2 *Adding the new file format Q5COST:*

Here the C++ binding for the Q5COST library have been used to implement a Q5COST reader in Open Babel. The next files from the Q5COST distribution (PythonModule folder) has been adapted for the implementation:

PyCost.cxx

PyCost.h

Only the references to the python headers have been removed and the files have been renamed to q5cost.cpp and q5cost.h respectively.

All the consecutive steps have been done in the Open Babel folder.

1) *A file for the Q5COST format in src/formats has been created:*
   - From the example file exampleformat.cpp, which contains a heavily-annotated description of writing a new format, we have written the QC-ml/Q5COST format description file (see Appendix 1: OBq5costreader.cpp, OBq5cost.h):
     a. When reading in molecules (and thus performing a lot of molecular modifications) we have called OBMol::BeginModify() at the beginning

and OBMol::EndModify() at the end. This will ensure that perception routines do not run while we read in a molecule and are reset after our code finishes.

b. For the cases without bond information we have called OBMol::ConnectTheDots() and OBMol::PerceiveBondOrders() after OBMol::EndModify() to ensure bonds are assigned. We have considered for this case various input and output options that users can set from the command-line or GUI. We offer the following options:

   i. -as Call only OBMol::ConnectTheDots() (single bonds only)

   ii. -ab No bond perception

c. We have used generic data classes like OBMol and others as appropriate. The Q5COST format stores common data types (see figure 1) so we have added a subclass of OBGenericData for use by other formats and user code. This subclass OBq5cost can support all the data contemplated in the Q5COST data model (see Appendix 1: OBq5costreader.cpp).

2) *Changes needed in OBConversion Class*: Open Babel does not support the Fortran file descriptors schema used in the Q5COST library. When Open Babel read a file, only the C++ I/O stream is available in the format ReadMolecule method. There is no way to get the file name to open it using the required Q5COST procedure. To solve this it is needed to include the following line:

```
InFilename=filePath;
```

at the end of the function bool OBConversion::ReadFile (OBBase* pOb, std::string filePath) just before the Read in the src/obconversion.cpp source file (see Appendix 1: obconversion.cpp modification). The problem is that InFilename is a protected variable of the OBConversion class so it should be set inside it. After that, a simple call to the GetInFilename method provide the ReadMolecule method with the needed file name.

3) *Open Babel compilation*. Apart from the generic options described at the Open Babel site[x] some additional instructions are needed to compile with support for a new format:

a. Edit the Makefile.am file in src/formats. Modify the libformats_la_SOURCES and pkglib_LTLIBRARIES variables adding OBq5costreader.cpp and OBq5costreader.la respectively. Create the new variables OBq5costreader_la_SOURCES and OBq5costreader_la_LDFLAGS containing the needed source files and the linking flags. (See Appendix 2 src/formats/Makefile.am).

b. Run the commands “aclocal” and “automake” on the Open Babel folder to rebuild the Makefile.in files needed during the configure procedure.

c. The following script have been used to invoke the configure command:

```
#!/bin/bash
export F77=ifort
export Q5COSTPATH=/home/cesga/aurelio/Q5COST-openbabel/q5cost-0.9.6
./configure --prefix=$openbabel_dir
```

d. After that we have follow the standard procedure: make, make install

After all these steps we have a working Open Babel distribution with full Q5COST format support (see Figure 3).

```
aurelio@svgd ~/Q5COST-openbabel/reader> babel -Hq5cost
q5cost  Q5COST format
Read only.
b no bonds
s no multiple bonds


Specification at: http://abigrid.cineca.it/abigrid/the-docs-archive/q5cost/index_html
aurelio@svgd ~/Q5COST-openbabel/reader> babel -iq5cost test.q5 -opdb test.pdb
1 molecule converted
9 audit log messages
aurelio@svgd ~/Q5COST-openbabel/reader> babel -iq5cost test.q5 -oxyz test.xyz
1 molecule converted
7 audit log messages
aurelio@svgd ~/Q5COST-openbabel/reader>
```

Figure 3 Open Babel Q5COST support at the CESGA SVGD server

Also using Open Babel API it is possible to access all the data supported by the Q5COST format through OBGenericData subclass, OBq5cost. An example file have been written to test the final Open Babel distribution (see Figure 4 and Appendix 1: example.cpp: Open Babel API and Q5COST).

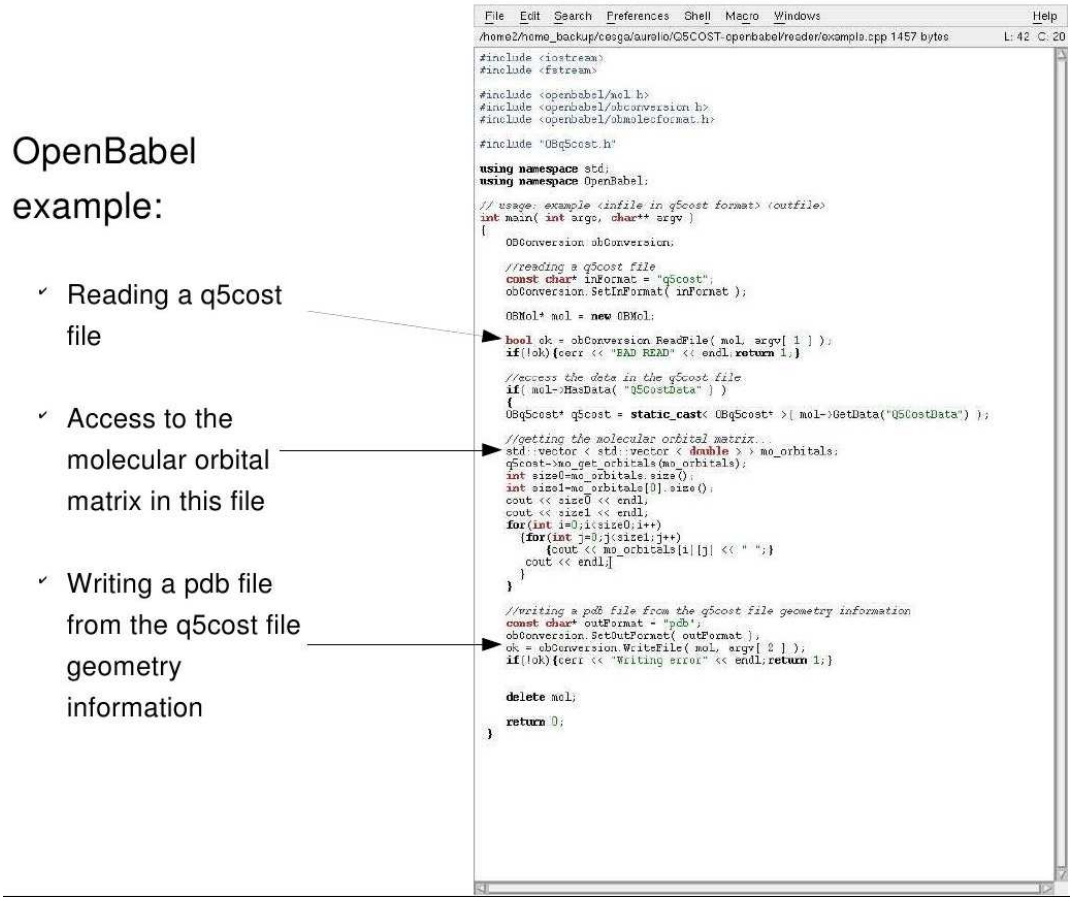

Figure 4 Open Babel API and Q5COST example

6 Future Work

The following subjects will be covered in the next future:

6.1 *Q5COST Open Babel Writer*

The Writer implementation is still in development. We are planning to support format interconversion using Open Babel at different levels, going from only geometry to specialized Quantum Chemistry data like atomic/molecular coefficients, needed for some specific formats (like Gamess, Gaussian …).

6.2 *Molekel, a specialized Quantum Chemistry Visualization tool.*

We will focus our effort upon the Molekel development as a specialized Quantum Chemistry Visualization tool, supporting the visualization of all the Q5COST data (from atomic orbitals to molecular orbitals and different properties). An “evaluate”

function will be developed for all the volumetric information and integrated on the OBGenericData subclass, OBq5cost.
Also this full Q5COST data support will open future developments in Quantum Chemistry visualization involving different magnitudes like the planned at the Q5COST WaveFunction domain. This support will start at the Molekel next release.

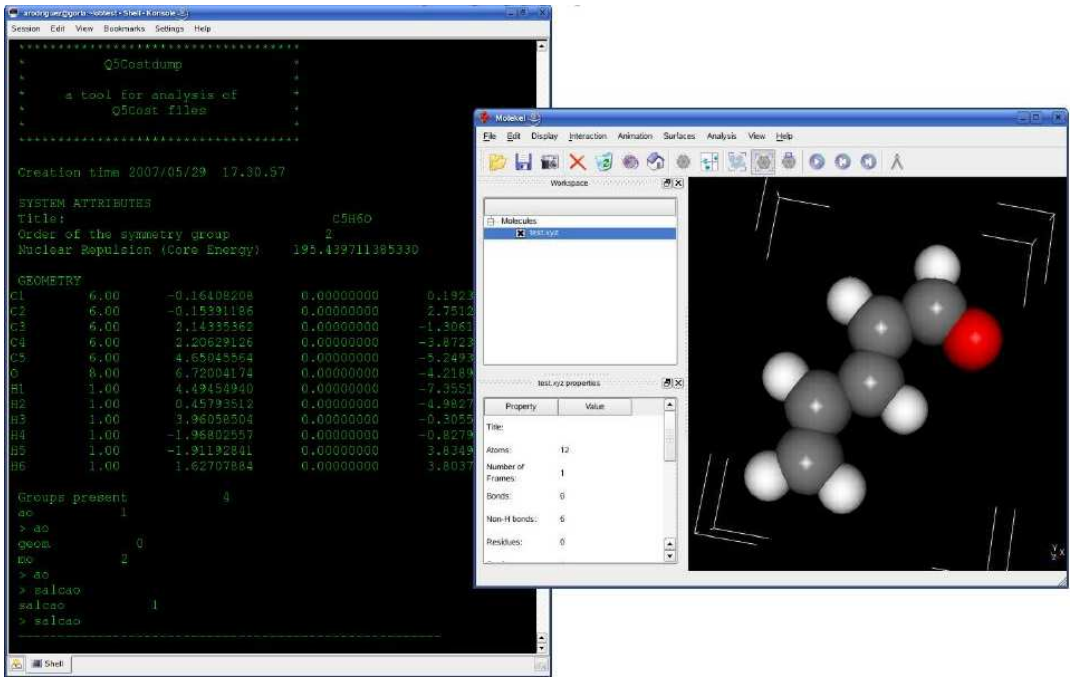

Figure 5 Q5COST MOLEKEL support

7 Appendix 1: Developed source codes and changes

OBq5costreader.cpp:

```
// Sample ObenBabel reader: implements a q5cost geometry reader

#include "q5cost.h"
// STD
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <cstring>
// reference: http://abigrid.cineca.it/abigrid/the-docs-archive/q5cost/index_html

#include <openbabel/obconversion.h>
#include <openbabel/obmolecformat.h>

#include "OBq5cost.h"

using namespace std;
using namespace OpenBabel;


namespace
{

    struct q5cost
    {
        int num_atoms;
        vector < vector < double > > coord;
        vector < string  > labels;
        vector < double > charges;

        //AO
        bool ao_ao_exists;
        string ao_coord_sys;
        int ao_sym_max;
        vector < vector < int > > ao_symmetry;
        vector < vector < vector < double > > > ao_contr;
        vector < vector < double > > ao_expo;
        vector < string > ao_user_tags;

        //MO
        bool mo_mo_exists;
        string mo_ao_ref_tag;
        string mo_basis_type;
        vector < string > mo_classification;
        vector < string > mo_labels;
        vector < vector < double > > mo_orbitals;
        vector < int > mo_num_orb_sym;
        vector < double > mo_occ_num;
        vector < double > mo_orb_energy;
        vector < int > mo_symmetry;
        vector < string > mo_user_tags;

        //WF (still not implemented ...)
    };

    bool ReadQ5Cost( q5cost& q5, string& in )
    {
        try
        {


           init();

           int file_id=file_open(in.c_str());

           //GEOMETRY
```

```
            if(geom_exists(file_id))
            {
            // number of atoms
            q5.num_atoms=geom_get_num_atoms(file_id);
            // atoms positions
            q5.coord=geom_get_coord(file_id,3,q5.num_atoms);
            // atoms labels
            q5.labels=geom_get_labels(file_id,q5.num_atoms);
            q5.charges=geom_get_charges(file_id,q5.num_atoms);
            }

            //AO
            string str_user_tag="ao"; //can be a problem f77 implementation
            q5.ao_ao_exists=ao_exists(file_id,str_user_tag);
            if(q5.ao_ao_exists)
            {
            q5.ao_coord_sys=ao_get_coord_sys(file_id,str_user_tag);
            q5.ao_sym_max=ao_get_sym_max(file_id,str_user_tag);
            int num_expo_max=ao_get_num_expo_max(file_id,str_user_tag);
            q5.ao_expo=ao_get_expo(file_id,num_expo_max,q5.num_atoms,str_user_tag);
            int num_contr_max=ao_get_num_contr_max(file_id,str_user_tag);
            q5.ao_contr=
            ao_get_contr(file_id,num_expo_max,num_contr_max,q5.num_atoms,str_user_tag);
            q5.ao_symmetry=
            ao_get_symmetry(file_id,num_contr_max,q5.num_atoms,str_user_tag);
            int size user tags=ao get num user tags(file id);
            q5.ao_user_tags=ao_get_user_tags(file_id,size_user_tags);

            }

            //MO
            str_user_tag="salcao";
            q5.mo_mo_exists=mo_exists(file_id,str_user_tag);
            if(q5.mo mo exists)
            {
            q5.mo_ao_ref_tag=mo_get_ao_ref_tag(file_id,str_user_tag);
            q5.mo_basis_type=mo_get_basis_type(file_id,str_user_tag);
            int num_orb_tot=mo_get_num_orb_tot(file_id,str_user_tag);
            q5.mo_classification=mo_get_classification(file_id,num_orb_tot,str_user_tag);
            q5.mo_labels=mo_get_labels(file_id,num_orb_tot,str_user_tag);
            int salcao_num_orb_tot=salcao_get_num_orb_tot(file_id,str_user_tag);
            q5.mo_orbitals=
            mo_get_matrix(file_id,num_orb_tot,salcao_num_orb_tot,str_user_tag);
            int num_sym=system_get_num_sym(file_id);
            q5.mo_num_orb_sym=mo_get_num_orb_sym(file_id,num_sym,str_user_tag);
            q5.mo_occ_num=mo_get_occ_num(file_id,num_orb_tot,str_user_tag);
            q5.mo_orb_energy=mo_get_orb_energy(file_id,num_orb_tot,str_user_tag);
            q5.mo_symmetry=mo_get_symmetry(file_id,num_orb_tot,str_user_tag);
            int size_user_tags=mo_get_num_user_tags(file_id);
            q5.mo_user_tags=mo_get_user_tags(file_id,size_user_tags);

            }

        }
        catch( const exception& )
        {
            return false;
        }

        return true;
    }
}


class OBQ5CostFormat : public OBMoleculeFormat

{

public:
    /// Co1nstructor: register 'q5cost' and "Q5COST" format.
    OBQ5CostFormat()
    {
        OpenBabel::OBConversion::RegisterFormat( "q5cost", this );
```

```
        OpenBabel::OBConversion::RegisterFormat( "Q5COST", this );
    }

    /// Return description.
    virtual const char* Description() //required
    {
        return
        "Q5COST format\n"
        "Read only.\n"
        "b no bonds\n"
        "s no multiple bonds\n\n";
    }

    /// Return a specification url
    virtual const char* SpecificationURL()
    {
        return "http://abigrid.cineca.it/abigrid/the-docs-archive/q5cost/index_html";
    }

    /// Return MIME type, NULL in this case.
    virtual const char* GetMIMEType() { return 0; };

      /// Return read/write flag: read only.
    virtual unsigned int Flags()
    {
        return READBINARY;
    };

    /// Skip to object: used for multi-object file formats.
///    virtual int SkipObjects( int n, OpenBabel::OBConversion* pConv ) { return 0; }

    /// OpenBabel calls the OBMoleculeFormat::ReadMolecule() passing a pre-created
       /// molecule instance and an OBConversion object instance which contains the
       /// input stream and and the input parameters (compute bonds etc.).
       /// The sequence of operations to implement for adding data into the passed
       /// OBMol instance are:
       /// -# get the input stream
       /// -# call OBMol::BeginModify()
       /// -# add atoms (and bonds if you do not want OpenBabel to compute the bonds)
       /// -# add custom data: custom data in the form of instances of classes derived
       ///         from OpenBabel::OBGenericData can be added on a per molecule
       ///         or per atom basis through the OBAtom::SetData() and OBMol::SetData()
       ///                   methods
       /// -# (optional) ask OpenBabel to compute bonds through OBMol::ConnectTheDots()
       /// -# call OBMol::EndModify()
       ///
       /// The method must return true if no error occured, false otherwise.
       /// Use the obErrorLog global variable to report errors
       ///  by invoking obErrorLog.ThrowError().

    virtual bool ReadMolecule( OBBase* pOb, OBConversion* pConv );

    /// Write: always returns false. Not implemented yet.
    virtual bool WriteMolecule( OpenBabel::OBBase* , OpenBabel::OBConversion* )
    {
        return false;
    }
};

//----------------------------------------------------------------------------

/// Global variable used to register the Q5Cost Format
OBQ5CostFormat theQ5CostFormat;


//----------------------------------------------------------------------------


//============================================================================

//----------------------------------------------------------------------------
bool OBQ5CostFormat::ReadMolecule( OBBase* pOb, OBConversion* pConv )
```

```
{
       // Get pointer to molecule
    OBMol* pmol = dynamic_cast< OBMol* >(pOb);
    if( pmol == 0 ) return false;

    // Get filename reference

    string ifs = pConv->GetInFilename();
    pmol->SetTitle(ifs.c_str());

    // Signal the start of the editing transaction.
    pmol->BeginModify();

    // Parse the input stream and use the OpenBabel API to populate the OBMol
    q5cost q5;
    if( !ReadQ5Cost( q5, ifs ) )
    {
        obErrorLog.ThrowError( __FUNCTION__, "Problems reading a q5cost file.",
                               obWarning);
        pmol->EndModify();
        return false;
    }

    // Set geometry to 3D
    pmol->SetDimension( 3 );
    // (optional) Reserve space for atoms
    pmol->ReserveAtoms( q5.num_atoms );
    for( int i = 0; i < q5.num_atoms; ++i )
    {
       // Create new atom
        OBAtom *atom = pmol->NewAtom();
        // Set type
        atom->SetType(q5.labels[i]);
        atom->SetAtomicNum(int(q5.charges[i]));
        // Set atom position
        atom->SetVector( q5.coord[ i ][ 0 ],
                         q5.coord[ i ][ 1 ],
                         q5.coord[ i ][ 2 ] );
    }

    // Read q5cost data different from geometry stuff
    OBq5cost* q5cost = new OBq5cost;

    //AO
    if(q5.ao_ao_exists)
    {
    q5cost->ao_set_coord_sys(q5.ao_coord_sys);
    q5cost->ao_set_sym_max(q5.ao_sym_max);
    q5cost->ao_set_symmetry(q5.ao_symmetry);
    q5cost->ao_set_contr(q5.ao_contr);
    q5cost->ao_set_expo(q5.ao_expo);
    q5cost->ao_set_user_tags(q5.ao_user_tags);
    }

    //MO
    if(q5.mo_mo_exists)
    {
    q5cost->mo_set_ao_ref_tag(q5.mo_ao_ref_tag);
    q5cost->mo_set_basis_type(q5.mo_basis_type);
    q5cost->mo_set_class(q5.mo_classification);
    q5cost->mo_set_labels(q5.mo_labels);
    q5cost->mo_set_orbitals(q5.mo_orbitals);
    q5cost->mo_set_num_orb_sym(q5.mo_num_orb_sym);
    q5cost->mo_set_occ_num(q5.mo_occ_num);
    q5cost->mo_set_orb_energy(q5.mo_orb_energy);
    q5cost->mo_set_symmetry(q5.mo_symmetry);
    q5cost->mo_set_user_tags(q5.mo_user_tags);
    }

    // Add custom data to molecule
    if(q5.ao_ao_exists || q5.mo_mo_exists)pmol->SetData( q5cost );
```

```
    if( !pConv->IsOption( "b", OBConversion::INOPTIONS ) ) pmol->ConnectTheDots();
    if (!pConv->IsOption( "s", OBConversion::INOPTIONS )
        && !pConv->IsOption( "b", OBConversion::INOPTIONS ) )
    {
        pmol->PerceiveBondOrders();
    }
    pmol->EndModify();
    pConv->SetOneObjectOnly(true);

    return true;
}
```

OBq5cost.h:

```
#include <openbabel/obmolecformat.h>
#include <openbabel/generic.h>
#include <vector>
#include <algorithm>
#include <limits>
#include <cassert>
#include <string>

/// Class to store q5cost generic data
class OBq5cost : public OpenBabel::OBGenericData
{
public:
    /// Constructor assigns the values of type and attr protected data
    /// This values will be accessed through the GetDataType, HasData methods.
    OBq5cost() : OpenBabel::OBGenericData()
    {
        _type = OpenBabel::OBGenericDataType::CustomData0;
        _attr = "Q5CostData";
    }

    //fill in data functions...

    //AO

    void ao_set_coord_sys(std::string &ao_coord_sys)
    {ao_coord_sys_=ao_coord_sys;}
    void ao_set_sym_max(int ao_sym_max)
    {ao_sym_max_=ao_sym_max;}
    void ao_set_symmetry(std::vector < std::vector < int > > &ao_symmetry)
    {ao_symmetry_=ao_symmetry;}
    void ao_set_contr(std::vector < std::vector < std::vector < double > > > &ao_contr)
    {ao_contr_=ao_contr;}
    void ao_set_expo(std::vector < std::vector < double > > &ao_expo)
    {ao_expo_=ao_expo;}
    void ao_set_user_tags(std::vector < std::string > &ao_user_tags)
    {ao_user_tags_=ao_user_tags;}

    //MO
    void mo_set_ao_ref_tag(std::string &mo_ao_ref_tag)
    {mo_ao_ref_tag_=mo_ao_ref_tag;}
    void mo_set_basis_type(std::string &mo_basis_type)
    {mo_basis_type_=mo_basis_type;}
    void mo_set_class(std::vector < std::string > &mo_classification)
    {mo_classification_=mo_classification;}
    void mo_set_labels(std::vector < std::string > &mo_labels)
    {mo_labels_=mo_labels;}
    void mo_set_orbitals(std::vector < std::vector < double > > &mo_orbitals)
    {mo_orbitals_=mo_orbitals;}
    void mo_set_num_orb_sym(std::vector < int > &mo_num_orb_sym)
    {mo_num_orb_sym_=mo_num_orb_sym;}
    void mo_set_occ_num(std::vector < double > &mo_occ_num)
    {mo_occ_num_=mo_occ_num;}
    void mo_set_orb_energy(std::vector < double > &mo_orb_energy)
    {mo_orb_energy_=mo_orb_energy;}
    void mo_set_symmetry(std::vector < int > &mo_symmetry)
    {mo_symmetry_=mo_symmetry;}
    void mo_set_user_tags(std::vector < std::string > &mo_user_tags)
    {mo_user_tags_=mo_user_tags;}


    //get data back functions...

    //AO

    void ao_get_coord_sys(std::string &ao_coord_sys)
    {ao_coord_sys=ao_coord_sys_;}
    void ao_get_sym_max(int ao_sym_max)
    {ao_sym_max=ao_sym_max_;}
    void ao_get_symmetry(std::vector < std::vector < int > > &ao_symmetry)
    {ao_symmetry=ao_symmetry_;}
    void ao_get_contr(std::vector < std::vector < std::vector < double > > > &ao_contr)
    {ao_contr=ao_contr_;}
```

```
    void ao_get_expo(std::vector < std::vector < double > > &ao_expo)
    {ao_expo=ao_expo_;}
    void ao_get_user_tags(std::vector < std::string > &ao_user_tags)
    {ao_user_tags=ao_user_tags_;}

    //MO
    void mo_get_ao_ref_tag(std::string &mo_ao_ref_tag)
    {mo_ao_ref_tag=mo_ao_ref_tag_;}
    void mo_get_basis_type(std::string &mo_basis_type)
    {mo_basis_type=mo_basis_type_;}
    void mo_get_class(std::vector < std::string > &mo_classification)
    {mo_classification=mo_classification_;}
    void mo_get_labels(std::vector < std::string > &mo_labels)
    {mo_labels=mo_labels_;}
    void mo_get_orbitals(std::vector < std::vector < double > > &mo_orbitals)
    {mo_orbitals=mo_orbitals_;}
    void mo_get_num_orb_sym(std::vector < int > &mo_num_orb_sym)
    {mo_num_orb_sym=mo_num_orb_sym_;}
    void mo_get_occ_num(std::vector < double > &mo_occ_num)
    {mo_occ_num=mo_occ_num_;}
    void mo_get_orb_energy(std::vector < double > &mo_orb_energy)
    {mo_orb_energy=mo_orb_energy_;}
    void mo_get_symmetry(std::vector < int > &mo_symmetry)
    {mo_symmetry=mo_symmetry_;}
    void mo_get_user_tags(std::vector < std::string > &mo_user_tags)
    {mo user tags=mo user tags ;}

private:
    //check Q5cost 0.9.6 Documentation about data supported
    //http://abigrid.cineca.it/the-docs-archive/q5cost/q5cost.html

    //GEOMETRY:already supported by openbabel std OBAtom

    //AO
    std::string ao_coord_sys_;
    int ao_sym_max_;
    std::vector < std::vector < int > > ao_symmetry_;
    std::vector < std::vector < std::vector < double > > > ao_contr_;
    std::vector < std::vector < double > > ao_expo_;
    std::vector < std::string > ao_user_tags_;

    //MO
    std::string mo_ao_ref_tag_;
    std::string mo_basis_type_;
    std::vector < std::string > mo_classification_;
    std::vector < std::string > mo_labels_;
    std::vector < std::vector < double > > mo_orbitals_;
    std::vector < int > mo_num_orb_sym_;
    std::vector < double > mo_occ_num_;
    std::vector < double > mo_orb_energy_;
    std::vector < int > mo_symmetry_;
    std::vector < std::string > mo_user_tags_;

    //WF (still not implemented ...)

};
```

src/obconversion.cpp modification:

```
bool  OBConversion::ReadFile(OBBase* pOb, std::string filePath)
 {
   if(!pInFormat) return false;

   // if we have an old stream, free this first before creating a new one
   if (pInStream && NeedToFreeInStream) {
     delete pInStream;
   }

   ifstream *ifs = new ifstream;
   NeedToFreeInStream = true; // make sure we free this
   ios_base::openmode imode =
     pInFormat->Flags() & READBINARY ? ios_base::in|ios_base::binary : ios_base::in;

   ifs->open(filePath.c_str(),imode);
   if(!ifs || !ifs->good())
     {
       obErrorLog.ThrowError(__FUNCTION__,"Cannot read from " + filePath, obError);
       return false;
     }
   InFilename=filePath;

   return Read(pOb,ifs);
 }
```

example.cpp: Open Babel API and Q5COST

```
#include <iostream>
#include <fstream>

#include <openbabel/mol.h>
#include <openbabel/obconversion.h>
#include <openbabel/obmolecformat.h>

#include "OBq5cost.h"

using namespace std;
using namespace OpenBabel;

// usage: example <infile in q5cost format> <outfile>
int main( int argc, char** argv )
{
    OBConversion obConversion;

    //reading a q5cost file
    const char* inFormat = "q5cost";
    obConversion.SetInFormat( inFormat );

    OBMol* mol = new OBMol;

    bool ok = obConversion.ReadFile( mol, argv[ 1 ] );
    if(!ok){cerr << "BAD READ" << endl;return 1;}

    //access the data in the q5cost file
    if( mol->HasData( "Q5CostData" ) )
    {
    OBq5cost* q5cost = static_cast< OBq5cost* >( mol->GetData("Q5CostData") );

    //getting the molecular orbital matrix...
    std::vector < std::vector < double > > mo_orbitals;
    q5cost->mo_get_orbitals(mo_orbitals);
    int size0=mo_orbitals.size();
    int size1=mo_orbitals[0].size();
    cout << size0 << endl;
    cout << size1 << endl;
    for(int i=0;i<size0;i++)
      {for(int j=0;j<size1;j++)
          {cout << mo_orbitals[i][j] << " ";}
       cout << endl;
      }
    }

    //writing a pdb file from the q5cost file geometry information
    const char* outFormat = "pdb";
    obConversion.SetOutFormat( outFormat );
    ok = obConversion.WriteFile( mol, argv[ 2 ] );
    if(!ok){cerr << "Writing error" << endl;return 1;}


    delete mol;

    return 0;
 }
```

8 Appendix 2: Open Babel compilation. Changes needed.

src/formats/Makefile.am:

```
INCLUDES = -I$(top_builddir)/include -I$(top_srcdir)/data -I$(top_srcdir)/include

EXTRA_DIST = exampleformat.cpp
SUBDIRS =

if BUILD_XML
SUBDIRS += xml
endif

if BUILD_INCHI
SUBDIRS += inchi
endif

if BUILD_PCH
BUILT_SOURCES = all.h.gch
PCHFLAGS=-Winvalid-pch -x c++-header $(CPPFLAGS) $(INCLUDES)
all.h.gch: all.h Makefile \
        $(srcdir)/../../include/openbabel/babelconfig.h \
  $(srcdir)/../../include/openbabel/base.h \
        $(srcdir)/../../include/openbabel/generic.h \
        $(srcdir)/../../include/openbabel/mol.h \
        $(srcdir)/../../include/openbabel/oberror.h \
        $(srcdir)/../../include/openbabel/obconversion.h \
        $(srcdir)/../../include/openbabel/obmolecformat.h
        rm -f $@
        $(CXX) $(PCHFLAGS) $<
AM_CXXFLAGS= -include $(top_srcdir)/src/formats/all.h
clean-local:
        -rm -f all.h.gch
endif

# Unfortunately there are a few formats which still cannot compile shared
# modules (e.g., Cygwin). So we need to keep a duplicate target for "libformats"

if !BUILD_SHARED
noinst_LTLIBRARIES = libformats.la
if BUILD_INCHI
libformats_la_LIBADD = inchi/libinchi.la
else
libformats_la_LIBADD = -linchi
endif
libformats_la_SOURCES = \
        APIInterface.cpp obmolecformat.cpp \
        CSRformat.cpp PQSformat.cpp alchemyformat.cpp \
        acrformat.cpp \
        amberformat.cpp balstformat.cpp bgfformat.cpp boxformat.cpp \
        cacaoformat.cpp cacheformat.cpp carformat.cpp cccformat.cpp \
        chem3dformat.cpp chemdrawct.cpp chemdrawcdx.cpp \
        chemtoolformat.cpp cifformat.cpp \
        copyformat.cpp crkformat.cpp cssrformat.cpp \
        dmolformat.cpp fastsearchformat.cpp fastaformat.cpp \
        fchkformat.cpp \
        featformat.cpp fhformat.cpp fingerprintformat.cpp \
        freefracformat.cpp gamessformat.cpp gaussformat.cpp \
        ghemicalformat.cpp gromos96format.cpp hinformat.cpp \
        getinchi.cpp \
        inchiformat.cpp jaguarformat.cpp mdlformat.cpp mmodformat.cpp \
        mol2format.cpp molreport.cpp mopacformat.cpp \
        mpdformat.cpp mpqcformat.cpp nwchemformat.cpp pcmodelformat.cpp \
        pdbformat.cpp povrayformat.cpp qchemformat.cpp reportformat.cpp \
        rxnformat.cpp shelxformat.cpp smilesformat.cpp thermoformat.cpp \
        tinkerformat.cpp titleformat.cpp \
        turbomoleformat.cpp unichemformat.cpp viewmolformat.cpp \
        xedformat.cpp xyzformat.cpp yasaraformat.cpp zindoformat.cpp \
        OBq5costreader.cpp
```

```
else BUILD_SHARED

pkglib_LTLIBRARIES = \
       APIInterface.la \
       CSRformat.la PQSformat.la alchemyformat.la \
       acrformat.la \
       amberformat.la balstformat.la bgfformat.la boxformat.la \
       cacaoformat.la cacheformat.la cansmilesformat.la \
       carformat.la cccformat.la \
       chem3dformat.la chemdrawctformat.la chemdrawcdxformat.la \
       chemtoolformat.la cifformat.la \
       copyformat.la crkformat.la cssrformat.la \
       dmolformat.la fastsearchformat.la fastaformat.la \
       fchkformat.la \
       featformat.la fhformat.la fingerprintformat.la \
       freefracformat.la gamessformat.la gaussformat.la ghemicalformat.la \
       gromos96format.la hinformat.la inchiformat.la \
       jaguarformat.la mdlformat.la mmodformat.la mpdformat.la \
       mol2format.la molreportformat.la mopacformat.la mpqcformat.la \
       nwchemformat.la pcmodelformat.la \
       pdbformat.la povrayformat.la qchemformat.la reportformat.la \
       rxnformat.la shelxformat.la smilesformat.la thermoformat.la \
       tinkerformat.la titleformat.la \
       turbomoleformat.la unichemformat.la viewmolformat.la \
       xedformat.la xyzformat.la yasaraformat.la zindoformat.la \
       OBq5costreader.la

APIInterface_la_SOURCES = APIInterface.cpp
APIInterface_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

CSRformat_la_SOURCES = CSRformat.cpp obmolecformat.cpp
CSRformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

PQSformat_la_SOURCES = PQSformat.cpp obmolecformat.cpp
PQSformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

acrformat_la_SOURCES = acrformat.cpp obmolecformat.cpp
acrformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

alchemyformat_la_SOURCES = alchemyformat.cpp obmolecformat.cpp
alchemyformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

amberformat_la_SOURCES = amberformat.cpp obmolecformat.cpp
amberformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

balstformat_la_SOURCES = balstformat.cpp obmolecformat.cpp
balstformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

bgfformat_la_SOURCES = bgfformat.cpp obmolecformat.cpp
bgfformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

boxformat_la_SOURCES = boxformat.cpp obmolecformat.cpp
boxformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

cacaoformat_la_SOURCES = cacaoformat.cpp obmolecformat.cpp
cacaoformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

cacheformat_la_SOURCES = cacheformat.cpp obmolecformat.cpp
cacheformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

cansmilesformat_la_SOURCES = cansmilesformat.cpp obmolecformat.cpp
cansmilesformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

carformat_la_SOURCES = carformat.cpp obmolecformat.cpp
carformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

cccformat_la_SOURCES = cccformat.cpp obmolecformat.cpp
cccformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

chem3dformat_la_SOURCES = chem3dformat.cpp obmolecformat.cpp
chem3dformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

chemdrawctformat_la_SOURCES = chemdrawct.cpp obmolecformat.cpp
```

```
chemdrawctformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

chemdrawcdxformat_la_SOURCES = chemdrawcdx.cpp obmolecformat.cpp
chemdrawcdxformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

chemtoolformat_la_SOURCES = chemtoolformat.cpp obmolecformat.cpp
chemtoolformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

cifformat_la_SOURCES = cifformat.cpp obmolecformat.cpp
cifformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

copyformat_la_SOURCES = copyformat.cpp
copyformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

crkformat_la_SOURCES = crkformat.cpp obmolecformat.cpp
crkformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

cssrformat_la_SOURCES = cssrformat.cpp obmolecformat.cpp
cssrformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

dmolformat_la_SOURCES = dmolformat.cpp obmolecformat.cpp
dmolformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

fastsearchformat_la_SOURCES = fastsearchformat.cpp
fastsearchformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

fastaformat_la_SOURCES = fastaformat.cpp obmolecformat.cpp
fastaformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

fchkformat la SOURCES = fchkformat.cpp obmolecformat.cpp
fchkformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

featformat_la_SOURCES = featformat.cpp obmolecformat.cpp
featformat la LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

fhformat_la_SOURCES = fhformat.cpp obmolecformat.cpp
fhformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

fingerprintformat_la_SOURCES = fingerprintformat.cpp obmolecformat.cpp
fingerprintformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

freefracformat_la_SOURCES = freefracformat.cpp obmolecformat.cpp
freefracformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

gamessformat_la_SOURCES = gamessformat.cpp obmolecformat.cpp
gamessformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

gaussformat_la_SOURCES = gaussformat.cpp obmolecformat.cpp
gaussformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

ghemicalformat_la_SOURCES = ghemicalformat.cpp obmolecformat.cpp
ghemicalformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

gromos96format_la_SOURCES = gromos96format.cpp obmolecformat.cpp
gromos96format_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

hinformat_la_SOURCES = hinformat.cpp obmolecformat.cpp
hinformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

inchiformat_la_SOURCES = inchiformat.cpp obmolecformat.cpp getinchi.cpp
if BUILD_INCHI
inchiformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la
inchi/libinchi.la
else
inchiformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la
inchiformat_la_LIBADD = -linchi
endif

jaguarformat_la_SOURCES = jaguarformat.cpp obmolecformat.cpp
jaguarformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

mdlformat_la_SOURCES = mdlformat.cpp obmolecformat.cpp
mdlformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la
```

```
mmodformat_la_SOURCES = mmodformat.cpp obmolecformat.cpp
mmodformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

mol2format_la_SOURCES = mol2format.cpp obmolecformat.cpp
mol2format_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

molreportformat_la_SOURCES = molreport.cpp obmolecformat.cpp
molreportformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

mopacformat_la_SOURCES = mopacformat.cpp obmolecformat.cpp
mopacformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

mpdformat_la_SOURCES = mpdformat.cpp obmolecformat.cpp
mpdformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

mpqcformat_la_SOURCES = mpqcformat.cpp obmolecformat.cpp
mpqcformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

nwchemformat_la_SOURCES = nwchemformat.cpp obmolecformat.cpp
nwchemformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

pcmodelformat_la_SOURCES = pcmodelformat.cpp obmolecformat.cpp
pcmodelformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

pdbformat la SOURCES = pdbformat.cpp obmolecformat.cpp
pdbformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

povrayformat_la_SOURCES = povrayformat.cpp obmolecformat.cpp
povrayformat la LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

qchemformat_la_SOURCES = qchemformat.cpp obmolecformat.cpp
qchemformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

reportformat_la_SOURCES = reportformat.cpp obmolecformat.cpp
reportformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

rxnformat_la_SOURCES = rxnformat.cpp
rxnformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

shelxformat_la_SOURCES = shelxformat.cpp obmolecformat.cpp
shelxformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

smilesformat_la_SOURCES = smilesformat.cpp obmolecformat.cpp
smilesformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

thermoformat_la_SOURCES = thermoformat.cpp obmolecformat.cpp
thermoformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

tinkerformat_la_SOURCES = tinkerformat.cpp obmolecformat.cpp
tinkerformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

titleformat_la_SOURCES = titleformat.cpp obmolecformat.cpp
titleformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

turbomoleformat_la_SOURCES = turbomoleformat.cpp obmolecformat.cpp
turbomoleformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

unichemformat_la_SOURCES = unichemformat.cpp obmolecformat.cpp
unichemformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

viewmolformat_la_SOURCES = viewmolformat.cpp obmolecformat.cpp
viewmolformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

xedformat_la_SOURCES = xedformat.cpp obmolecformat.cpp
xedformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

xyzformat_la_SOURCES = xyzformat.cpp obmolecformat.cpp
xyzformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

yasaraformat_la_SOURCES = yasaraformat.cpp obmolecformat.cpp
yasaraformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la
```

```
zindoformat_la_SOURCES = zindoformat.cpp obmolecformat.cpp
zindoformat_la_LDFLAGS = -module -avoid-version -no-undefined ../libopenbabel.la

OBq5costreader_la_SOURCES = OBq5costreader.cpp q5cost.cpp obmolecformat.cpp
OBq5costreader_la_LDFLAGS = -module -avoid-version -no-undefined -L$(Q5COSTPATH)/lib
                            -lq5cost ../libopenbabel.la

endif
```

[i] http://abigrid.cineca.it/abigrid/
[ii] Action D37: Grid Computing in Chemistry: GRIDCHEM
http://www.cost.esf.org/index.php?id=189&action_number=d37
[iii] C. Angeli, G. L. Bendazzoli, S. Borini, R. Cimiraglia, A. Emerson, S. Evangelisti, D. Maynau, A. Monari, E. Rossi, J. Sanchez-Marin, P. G. Szalay, A. Tajti, "The Problem of Interoperability: A Common Data Format for Quantum Chemistry Codes", Int. Journ. Quant. Chem., 107, 2082-2091 (2007).
S. Borini, A. Monari, E. Rossi, A. Tajti, C. Angeli, G. L. Bendazzoli, R. Cimiraglia, A. Emerson, S. Evangelisti, D. Maynau, J. Sanchez-Marin, P. G. Szalay, "FORTRAN Interface for Code Interoperability in Quantum Chemistry: The Q5Cost Library", Journ. Chem. Inf. Modell., 47(3), 1271-1277 (2007).
[iv] http://abigrid.cineca.it/the-docs-archive/q5cost/q5cost.html
[v] Rajarshi Guha, Michael T. Howard, Geoffrey R. Hutchison, Peter Murray-Rust, Henry Rzepa, Christoph Steinbeck, Jörg Wegner, and Egon L. Willighagen, "The Blue Obelisk - Interoperability in Chemical Informatics", J. Chem. Inf. Model., 46, 991-998 (2006).
[vi] http://openbabel.sourceforge.net/wiki/Category:Formats
[vii] http://openbabel.sourceforge.net/api/2.1.0/main.shtml
[viii] http://abigrid.cineca.it/abigrid/download/download-q5cost-detailed-instructions
[ix] http://openbabel.sourceforge.net/wiki/Get_Open_Babel
[x] http://openbabel.sourceforge.net/wiki/Install_%28source_code%29