Q5Cost Format and Library: A Tutorial About the Common Format for Quantum Chemistry Interoperability

A. Monari[1], A. Scemama[2], S. Evangelisti[2] and E. Rossi[3*]

[1] *Dipartimento di Chimica Fisica e Inorganica, Università di Bologna, Bologna, Italy*
[2] *Laboratoire de Chimie et Physique Quantiques, UMR 5626, Université de Toulouse et CNRS 118, Toulouse, France*
[3] *CINECA, Casalecchio di Reno, Bologna, Italy*

Lecture given at the Joint EU-IndiaGrid/CompChem Grid Tutorial on Chemical and Material Science Applications Trieste, 15-18 September 2008

LNS0924007

*e.rossi@cineca.it

Abstract

A user guide concerning the use of the Q5Cost format and library is presented. The present tutorial is aimed to the Quantum Chemistry developers who wish to exploit the Q5Cost capabilities in their own code. Q5Cost is a data format for interchange and interoperability, specifically designed for Quantum Chemistry programs, and was proposed within a COST activity. A brief recall of the format and library structure is presented. The installation procedure, the naming convention and the coding strategy are fully illustrated in a tutorial style. Many examples are provided and discussed in order to facilitate the usage for the final programmer. Several utilities have been delivered for the use of the developers and the users of the library, as well as various converters for well-known Quantum Chemistry codes.

Contents

1 Introduction

The present article contains a tutorial for the use of a library, called Q5Cost, for the management of data produced by a generic ab-initio Quantum Chemistry (QC) code and its conversion from and to a common data format for the Quantum Chemistry, also called Q5Cost.

The Q5Cost data format has been defined with the aim of making code interoperability easier in the scientific area of Quantum Chemistry. This is the outcome of an activity carried out within the COST network, supporting cooperation among researchers in Europe: the former D23 "Metache" action, now concluded, and the D37 "GridChem" action, started in July 2006 [1].

The first problem to face when integrating different QC codes in a common workflow is the lack of a standard and the different data formats adopted by each code. Our suggestion is to adopt such a standard, specifically designed for interchange, and to connect each code in the set through a converter. Of course not to invent "yet another format", we strongly tried to design a format as general and flexible as possible and to coordinate ourselves with other similar initiatives in the QC context.

The Q5Cost format and library is based on and takes advantage of HDF5 (Hierarchical Data Format) [2]. The definition of the data model, the description of the library and the last uptake of the activity has been reported elsewhere [3, 4, 5, 6].

1.1 The intended audience

This tutorial is designed for final users, researchers or programmers that need to manage the data files produced by a QC calculation in order to increase the interoperability of the program itself. This goal can be reached by writing a specific translation program (wrapper) for each QC code towards a common data format. If the common data format is Q5Cost, the routines of the Q5Cost library can facilitate this task.

The same goal can be reached also by including calls to the library in the specific QC code, if this seams more suitable to the user.

1.2 The data format

The structure of the Q5Cost data format has been fully described elsewhere [3, 4, 5, 6], as well as its connections with the inherent HDF5 structure [2].

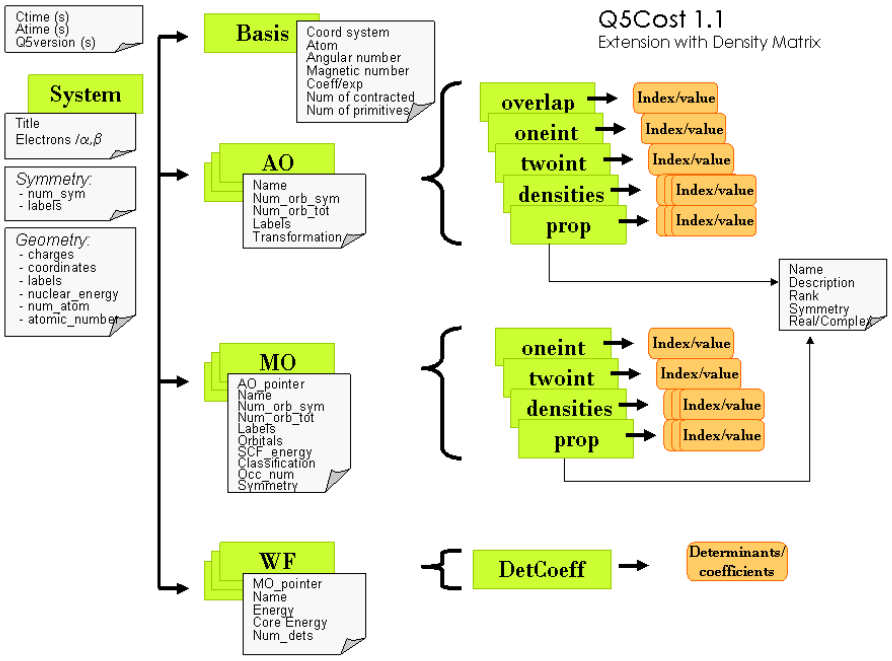

Figure 1: Schema of the Q5Cost data format.

Here we just want to briefly recall the main features that are useful for understanding how to apply the library.

Q5Cost is a common interchange format for data coming from ab initio quantum chemistry calculations. In this context ab initio refers to methods that are based on a wave function rather than on an electronic density. In particular, the Density Functional Theory (DFT) methods at the moment are not supported by our format. Extension to the DFT context, as well as to Quantum Dynamics context are under development.

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 dark gray boxes are containers (groups in HDF5 terminology) and the light gray boxes are data (data sets), while the metadata (attributes) are listed in the white labels.

Data described by the Q5Cost format belong to two different categories:

1. **Data:** The large binary quantities used in quantum chemistry for representing integrals, properties and wave functions. They can be stored using matrices with an arbitrary number of indices (rank-n arrays), scale aggressively with the system size, and are normally accessed with a "chunked" approach (i.e., using well-defined blocks of data).

2. **Metadata:** Simple and small pieces of data that describe and better define the previous data. They represent well-known chemical entities like nuclear energy, molecular orbital labels, molecular symmetry and can be stored as scalars, vectors or matrices. For example, the nuclear repulsion energy is a floating point scalar, molecular orbitals coefficients 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.

All the chemical objects described in the Q5Cost data format are related within a hierarchical structure and a logical containment relation. A first (root) container, named System, represents the molecular system as defined by its structural data (chemical composition and spatial geometry). All the general metadata can be associated to this container. A system can contain several Domains. The role of the Domain is to group together entities whose indices conceptually refer to the same kind of functions. In the present version three Domains have been defined:

- **Atomic Orbital** (AO): Refers to data defined on the AO basis, overlap, one-electron integrals, two-electron integrals, Density matrix and the generic property, i.e. any other property that can be described on the AO basis (dipole moment integrals, for example). This domain contains also the definition of the Basis Set.

- **Molecular Orbital** (MO): Refers to the data defined on the MO basis, one- and two-electron integrals, Density matrix and the generic property. This domain also contains the transformation matrix needed to define the MO on the AO basis.

- **Wave Function** (WF): Refers to the definition of the wave function.

1.3 The Library

The library is organized as a collection of many routines (more than 100 are available at the moment). The routine names are chemically significant and strictly recall the function of the routine itself. Each routine belongs to a different class, the classification is made according to the references of the routines to the domains and/or to specific objects in a given domain.

At present different classes of routines are available:

```
Q5Cost_init     Routines for housekeeping and file management
Q5Cost_deinit
Q5Cost_file_
Q5Cost_Basis_ Basis set
Q5Cost_System_ General high level information
  Q5Cost_Sys_Geometry_ Geometry of the molecule
  Q5Cost_Sys_Symmetry_ Space symmetry of the molecule
Q5Cost_AO_ Atomic Orbitals
  Q5Cost_AOOneInt_ Atomic Orbitals: one electron integrals
  Q5Cost_AOTwoInt_ Atomic Orbitals: one electron integrals
  Q5Cost_AOOverlap_ Atomic Orbitals: overlap integrals
  Q5Cost_AODensity_ Density Matrix on the AO orbitals
    Q5Cost_AODensityOne_ Density Matrix on the AO orbitals: One-body
    Q5Cost_AODensityTwo_ Density Matrix on the AO orbitals: Two-bodies
Q5Cost_MO_ Molecular Orbitals
  Q5Cost_MOOneInt_ Molecular Orbitals: one electron integrals
  Q5Cost_MOTwoInt_ Molecular Orbitals: two electrons integrals
  Q5Cost_MODensity_ Density Matrix on the MO orbitals
    Q5Cost_MODensityOne_ Density Matrix on the MO orbitals: One-body
    Q5Cost_MODensityTwo_ Density Matrix on the MO orbitals: Two-bodies
Q5Cost_Property_ Integrals for a generic operator
Q5Cost_WF_ Wave-function
  Q5Cost_WFDetCoef_  Wave-function: Determinant and coefficients
```

The naming convention depends on the type or routines, and follows mainly two schemes:

1. Routines working on the large QC datasets, for example:
   Q5Cost_AOOneInt_append

   - *Q5Cost*: is a common prefix

- *AOOneInt*: is the data object involved, in this case the one electron integrals on the AO basis
- *append*: is the operation to be performed, in this case an append (write) operation.

2. Routines working on the domains and on the metadata (small data) for example:
   Q5Cost_AO_set_num_orb_sym
   - *Q5Cost*: is a common prefix
   - *AO*: is the domain where the action will work
   - *set*: is the operation to be performed
   - *num_orb_sym*: is the data object affected by the operation

Also the formal parameters inside the routines follow a specific organisation. For example, let us consider the Q5Cost_AOOverlap_read routine:

```
Q5Cost_AOOverlap_read (file_id,offset,howmany,idx,value,error,
                       user_tag)
```

The first parameter is always the ID of the .q5 file, that is returned by an initial "file_create" or "file_open" routine. All the other parameters, up to "error", are input/output parameters and depends on the actual routine: in this case "offset" is the numeral of the first integral to be read from the file, "howmany" is the total number of integrals, "idx" is the place where to store the indices, "value" is the place where to store the values (with this routine you can read only integrals with indices). The "error" parameter returns the error condition of the routine. A return value of zero for the "error" parameter means the routine completed successfully. All routines in the library have the "error" parameter. The following parameters, if any, are optional and can be invoked by key. Usually they have a default value attached, so if you do not include them in the actual call, they assume a prespecified value. In this example the only optional parameter is "user_tag", a string to identify the actual AO domain these integrals refer to. If not given, the default "default_tag" string is used.

1.4 The organization of the .q5 file

The data written by the Q5Cost routines are stored into a file, usually identified by the ".q5" extension. This is a standard HDF5 file and can be

accessed using the HDF5 tools (h5dump, h5ls, etc., see HDF5 Utilities manual at the HDF web site [7]) even if other more specific tools are available in the Q5Cost distribution (q5dump, q5edit).

Each .q5 file contains a collection of data obtained through multiple QC calculations, but always refers to a single molecular specie, a single geometry and a single choice for the basis set functions. If the user needs more geometries (like in geometry optimization workflows) he has to organize the work on multiple .q5 files.

1.5 Utilities

A number of utilities have been setup to facilitate the use of the Q5Cost library and data format. Some of the utilities are addressed to the library developers, others are intended to be used by the final users: here only the latter will be discussed.

A first useful utility is an automatic configuration script to install the library on your system. It takes into account the specificity of the computing platform, like the compilers, the location of the libraries, and verifies if the needed underlying HDF5 layer is present. This configuration script is based on *autoconf* [8], an extensible package of macros that produces shell scripts to automatically configure software source code packages and guarantees the portability on different architectures.

Two other useful utilities for the final user are available to inspect the .q5 file without writing an ad-hoc program. Even if HDF5 is a binary file, it contains its own description and it is possible, using standard HDF5 instruments, to extract a complete description in a human readable form.

- **q5dump** is a simple tool that reports the content of a Q5Cost file, listing the data and the metadata contained in it.
- **q5edit** is a more complete and attractive tool. Using this Python interactive program, the user can navigate through the file, listing the content and modifying the data using his favorite text editor.

2 Working notes: A tutorial for the use of the Q5Cost library

The Q5Cost library and model are fully described in the home web site (http://www.q5cost.org) that can be used as the main reference for down-

loading the library and the tools, as well as to get up-to-date documentation and assistance.

2.1 Where to find the Q5Cost library

The last version of the library (1.0 at present) is downloadable from the Q5Cost web site following the download link.

Since the Q5Cost library is built on top of the HDF5 Fortran library, the first step is to install HDF5. At present the supported versions are HDF5 1.6.7 and 1.8.0 that can be downloaded from the official site of HDF5:
http://www.hdfgroup.org/HDF5/release/index.html

Some instructions for a quick installation of the HDF5 library can be found on the Q5Cost.org website too; follow them to download HDF5 and install it on your system. At this point you can proceed with the installation of Q5Cost. There are two ways to get the Q5Cost package:

1. get the compressed package from the web site
2. access to the CVS repository.

2.1.1 Get the compressed package

This is the simplest way for getting ready with Q5Cost. You can get detailed information about the installation of the web site:
http://abigrid.cineca.it/abigrid/download/download-q5cost-detailed-instructions.

After downloading the compressed package (.gz) on a Unix/Linux machine, you have to expand it using the **gunzip/tar** commands: this will create a new folder (Q5Cost-1.0) containing all you need for installing the library.

```
> gunzip q5cost-1.0.tar.gz
> tar -xvf  q5cost-1.0.tar
> cd q5cost-1.0
```

2.1.2 Access to the CVS repository

The Q5Cost library is available in a CVS repository in Toulouse. CVS (Concurrent Versions System) is a tool for version control and is used by the Q5Cost developers in order to assure a safe collaborative maintenance of the

code.

```
host = lpqsv11.ups-tlse.fr
cvs name = /misc/CVS_Repositories/cvs_cost
username (read only) = costch01
```

(ask the password if interested to q5cost_info@cineca.it).
The CVS repository contains the very last working version of the Q5Cost library; please note that this is not a stable and guaranteed version, you can use it at your own risk. If you want the last stable version, please download it from the official download site.

The costch01 username is allowed only to read from the repository. If you want to become a developer and get a username with write privileges, please, ask q5cost_info@cineca.it.

To access the CVS from your local machine you will have to follow these steps

1. connect to your local unix machine
2. enter the following commands

```
>export CVS_RSH=ssh
>cvs -d <username>@lpqsvll.ups-tlse.fr:
 /misc/CVS_Repositories/cvs_cost <command> <options>
```

For example, to copy the library in the local "q5cost" directory using the read only username, the commands are:

```
>export CVS_RSH=ssh
>cvs -d costch01@lpqsv11.ups-tlse.fr:
 /misc/CVS_Repositories/cvs_cost checkout q5qost-1.0
```

2.2 How to install the library and tools on your machine

Whichever way you choose to download the q5cost package, you now have a "q5cost" directory containing all you need for the installation. You can now start reading the 00_README file and run the configure script that will look for the required programming tools in your environment.

Before executing the configure script be sure to have the Fortran and C compilers, in addition to the HDF5 library. The configure script tries to locate all required modules using the *locate* command. If you want to give the exact location of the HDF5 library you can set the $LIBHDF5 environment variable before running the script. Also the compilers can be forced by setting some environmental variables: FC for the Fortran compiler, CC for the C compiler, CXX for the C++ compiler. You can obtain a complete description of the configure script with the "–help" option.

```
> export LIBHDF5=/.../libraries/hdf5/intel--10.1/lib
> ./configure --help
> ./configure
```

At this point some useful files have been created:

- config.log: This is a log of the configure run, useful for debugging the script
- config.status: This is a script that can be used to recreate the current configuration
- INSTRUCTIONS: Environment variables to be set for using the library
- Makefile: Script for compiling the library, the user tools, the test-suite and the Python modules.

To compile the library and the other tools using the Makefile enter the command:

```
> make all
```

If everything is fine you can check now your library setup running the test suite

```
> cd testsuite
> ./testsuite
```

If everything is fine you should obtain an output text confirming the correct execution of all programs in the testsuite. Do not worry about error messages, since some components of the suite check also the ability to recognize error conditions. Just verify that all tests passed, an "OK" string is printed for each of them.

At the end a summary is printed. It should appear like this one:

```
----- Final Output -------
  Test performed  :         146
  Test successful :         146
  Test failed     :           0
    First ones
  Test error      :           0
```

2.3 How to use it (wrappers or internal coding)

Q5Cost is a library and it is intended to be used within a program. Its objective is to help converting data format from/to a specific data format used by a QC program, by translating into Q5Cost, which is a common format for interchange. There are several bindings available: Fortran 90, Fortran 77, C, C++ and Python (Fortran 77 is not encouraged). So you can call the library routines from programs written in one of these languages.

There are mainly two different approaches that can be followed to interface a generic QC program with the Q5Cost data format:

1. You can write a "wrapper" for the QC program of interest: this is a separate program, coded in the language of choice, taking care of translating the data format from and to the program native format towards the Q5Cost model.

2. You can include the Q5Cost calls into the QC program for managing the read/write access to the data (internal coding).

The choice among the two approaches depends on the personal preferences and on the availability of the source code of the QC program. Both of them have pros and cons to be carefully considered before starting. Both approaches have been followed up to now by the partners of the project.

In any case, the first thing to do before starting to use the library is to correctly setup the working environment, by following the suggestions contained in the "INSTRUCTIONS" file in the root directory of Q5Cost. This file is created by the configure script and points out the correct variables to be defined for the actual environment:

1. Two environment variables have to be defined, pointing to the include directories of HDF5 and Q5Cost

```
> export INCHDF5=/.../hdf5/include
> export Q5COSTPATH=/.../q5cost-1.0
```

The definition can be included in the unix startup file (.bashrc, for example) in order to be always present.

2. When compiling and linking your program (wrapper or QC program with internal coding) you have to point the required libraries and includes. For example, if the variables reported above have been defined, a possible Makefile for your wrapper could be:

```
# Makefile for compiling and linking a program containing
# calls to the Q5Cost library
#
# define the following variables befor running the Makefile:
# - the include-HDF5 path (in $INCHDF5)
# - the Q5Cost root path (in $Q5CostPATH)
# - the compiler to be used (in $FC)
#
Fcomp  = $(FC)
FCFLAGS= -I$(Q5CostPATH)/include -I$(INCHDF5)
LDFLAGS= $(Q5CostPATH)/lib/Q5Cost.a
TARGETS=wrapper

.SUFFIXES: .f .o
.f.o: $*.f
        $(Fcomp) -c $(FCFLAGS) -o $@ $*.f
$(TARGET):  wrapper.o
$(Fcomp) -o $@ $@.o $(LDFLAGS)
```

The best way to start using the Q5Cost library is to try with some easy examples, which you can find in the "examples" directory under the Q5Cost root:

- file_create: creates a new, empty file named file.q5;
- file_open: open an existing file file.q5 and extract the creation date;
- geometry_create: creates a new file.q5 file and writes in it the geometry information, extracted from the "benzene.xyz" file;
- geometry_read: open the existing file file.q5 (created by the previous program) and extract geometry information from it.

All these examples are coded in F90, the most common language used in QC. Two of the examples are commented in the following.

2.4 Example 1: Creating or opening a .q5 file

In this example we create a .q5 file named file.q5

```
      PROGRAM FILE_CREATE
      use Q5Cost
      implicit none
      integer :: error ! The error parameters for the Q5Cost calls
                       ! if everything goes well error == 0
      character*64 :: file_name, ctime
      integer(hid_t) q5_id !q5_file identifier note use the hdf5 type
      file_name = 'file.q5'
! Initialize the library to be called at the beginning
! of each Q5Cost session
      call Q5Cost_init(error)
      if(error.ne.0) then
         write(6,*) 'Unable to Initialize Q5Cost'
         stop
      endif
!  And now creates the file named file_name
!  the file will be given the q5_id identifier
!  that will be used for any future reference
      call Q5Cost_file_create(file_name,q5_id,error)
      if(error.ne.0) then
         write(6,*) 'Unable to create Q5Cost'
         stop
      endif
!  You created the file let's get and print the creation time
      call Q5Cost_file_get_ctime(q5_id,ctime,error)
      if(error.ne.0) then
         write(6,*) 'Unable to get creation time'
         stop
      endif
      write(6,'(a4,a1,a2,a1,a2)') ctime(1:4),'/',ctime(5:6),
     $                            '/',ctime(7:8)
     write(6,'(a2,a1,a2,a1,a2)') ctime(10:11),':',ctime(12:13)
```

```
     $                            ,':',ctime(14:15)
!  Now we close the file
      call Q5Cost_file_close(q5_id,error)
      if(error.ne.0) then
         write(6,*) 'Unable to close Q5Cost'
         stop
      endif
!  and we deinitialize the Q5Cost library
      call Q5Cost_deinit(error)
      if(error.ne.0) then
         write(6,*) 'Unable to Initialize Q5Cost'
         stop
      endif

      END
```

```
use q5cost
```

This F90 statement is needed to access the Q5Cost module and is always required at the very beginning of all subroutines/functions/programs containing Q5Cost calls.

```
integer(hid_t) q5_id
```

This is the identifier of the .q5 file containing data in the Q5Cost format. This must be declared of type integer(hid_t) that is a specific type defined by hdf5. After the open call (create/open), this is the only reference to the .q5 file, used by all Q5Cost routines.

```
call Q5Cost_init(error)
call Q5Cost_deinit(error)
```

These routines initialize the Q5 environment. Each program using Q5Cost must"initialize" the environment at the beginning and "deinitialize" it at the end.

```
call Q5Cost_file_create(file_name,q5_id,error)
call Q5Cost_file_open  (file_name,q5_id,error)
call Q5Cost_file_close(q5_id,error)
```

Creates the file named file_name. If the file already exists you can open it (in R/W) with the Q5Cost_file_open routine: the file will be given the q5_id identifier that will be used for any future reference. When the file is no longer needed, it must be closed with the Q5Cost_file_close routine.

```
if(error.ne.0) then
   write(6,*) 'Unable to create Q5Cost'
   stop
endif
```

All Q5Cost routines contain an exit parameter (error) that can be tested to trap a possible error. If the routine was successful the return value is zero.

2.5 Example 2: Extracting information from a .q5 file

In this example we read the geometry information from a .q5 file named "file.q5".

```
      PROGRAM GEOMETRY_READ
!   Read the geometry from a Q5Cost file for a benzene molecule
      use Q5Cost
      implicit none
      integer :: error ! The error parameters for the Q5Cost calls
                       ! if everything goes well error == 0
      character*64 :: file_name, ctime
      integer(hid_t) :: q5_id !q5_file identifier note use the hdf5 type
      integer :: num_sym, num_alpha, num_beta, num_atoms,i
      character*3, allocatable :: label(:)
      real*8, allocatable :: charge(:), coord(:,:)
      file_name = 'file.q5'
! Initialize the library to be called at the beginning
! of each Q5Cost session
      call Q5Cost_init(error)
      if(error.ne.0) then
         write(6,*) 'Unable to Initialize Q5Cost'
         stop
      endif
!  And now creates the file named file_name
!  the file will be given the q5_id identifier
!  that will be used for any future reference
```

```
      call Q5Cost_file_open(file_name,q5_id,error)
      if(error.ne.0) then
         write(6,*) 'Unable to open Q5Cost'
         stop
      endif
! now read the geometrical parameters from the Q5Cost file
      call Q5Cost_Sys_Geometry_get_num_atom(q5_id,num_atoms,error)
      if(error.ne.0) then
         write(6,*) 'Unable to set number  of atoms'
         stop
      endif
! allocate the array storing the geometrical parameters
      allocate (charge (num_atoms) )
      allocate (label (num_atoms) )
      allocate (coord (3,num_atoms) )
      call Q5Cost_Sys_Geometry_get_charge(q5_id,charge,error)
      if(error.ne.0) then
         write(6,*) 'Unable to get charges  of atoms'
         stop
      endif
      call Q5Cost_Sys_Geometry_get_coord(q5_id,coord,error)
      if(error.ne.0) then
         write(6,*) 'Unable to get coordinates'
         stop
      endif
      call Q5Cost_Sys_Geometry_get_label(q5_id,label,error)
      if(error.ne.0) then
         write(6,*) 'Unable to get labels'
         stop
      endif
!  Write the parameters in a xyz format
      write(6,*) num_atoms
      write(6,*)
      do i = 1, num_atoms
        write(6,'(a4,4f16.8)') label(i),
     $       coord(1,i), coord(2,i), coord(3,i)
      enddo
!  Write the effective charges
```

```
      write(6,*)
      write(6,*) 'Charges'
      do i = 1, num_atoms
        write(6,'(a4,f6.4)') label(i), charge(i)
      enddo
!  Now we close the file
      call Q5Cost_file_close(q5_id,error)
      if(error.ne.0) then
         write(6,*) 'Unable to close Q5Cost'
         stop
      endif
!  and we deinitialize the Q5Cost library
      call Q5Cost_deinit(error)
      if(error.ne.0) then
         write(6,*) 'Unable to Initialize Q5Cost'
         stop
      endif
      END
```

```
Q5Cost_Sys_Geometry_get_num_atom(q5_id,num_atoms,error)
```

Access the geometry class in the .q5 file and get the number of atoms.

```
! allocate the array storing the geometrical parameters
      allocate (charge (num_atoms) )
      allocate (label (num_atoms) )
      allocate (coord (3,num_atoms) )
      call Q5Cost_Sys_Geometry_get_charge(q5_id,charge,error)
Q5Cost_Sys_Geometry_get_coord(q5_id,coord,error)
Q5Cost_Sys_Geometry_get_label(q5_id,label,error)
```

The nuclear effective charges ("charge(:)"), the Cartesian coordinates ("coord(3,:)") and the atom labels ("label(:)") are extracted from the file after dynamically allocating the corresponding arrays.

3 Staying tuned: Updates and help

The web site is the best place to stay tuned with the activity of the Q5Cost developing team. In the near future we plan to create a mailing list for

disseminating important news about the activity upgrades. The mailing list creation will be posted on the web site.

The library is well documented through a reference manual. It can be found on the web site as an html document. It is also included in the download package, into the "doc" folder as a pdf, txt and html document.

An official user help-desk is not yet in place. In the mean time you can send a mail to the Q5Cost developers if you need any advice:

`q5cost_info@cineca.it`

4 Acknowledgments

Support from COST in Chemistry action D37 is gratefully acknowledged. The French CNRS is also gratefully acknowledged for partial funding.

References

[1] The COST/D37 action:
http://www.cost.esf.org/domains_actions/cmst/Actions/
Grid_Computing_in_Chemistry

[2] HDF5 a general purpose library and file format for storing scientific data. Also available at http://hdf.ncsa.uiuc.edu/HDF5/ Specification of XML can be found at the site (http://www.w3.org/XML); A. Holmer, XML IE5 Programmers Reference, Wrox Press, Chicago, IL, USA, 1999

[3] E. Rossi, A. Emerson and S. Evangelisti, (2003). Lect. Notes Comput. Sci. 2658, 316-323

[4] S. Borini, A. Monari, E. Rossi, A. Tajti, C. Angeli, G.L. Bendazzoli, R. Cimiraglia, A. Emerson, S. Evangelisti, D. Maynau, J. Sanchez-Marin and P.G. Szalay, (2007). J. Chem. Inf. Model. 47, 1271-1277

[5] 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 and A. Tajti, (2007). Int. J. Quant. Chem. 107, 2082-2091

[6] A. Scemama, A. Monari, C. Angeli, S. Borini, S. Evangelisti and E. Rossi, (2008). Lect. Notes on Comp. Science **5072** 1094

[7] HDF5 Document Set: http://www.hdfgroup.org/HDF5/doc/index.html

[8] Gnu autoconf see: http://www.gnu.org/software/autoconf