PUPIL

Program for User Package Interfacing and Linking

User Manual

Version 1.2.1

March 15, 2009

Revision: User Manual v2.3

May 6, 2009

I. Disclaimer

PUPIL (Program for User Package Interface and Linking) is free software. You may redistribute it and/or modify it only under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

PUPIL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, including but not limited to any implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE or NON-INFRINGEMENT. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this software; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Neither the names of the Quantum Theory Project, EUETII, the University of Florida, the Universitat Politècnica de Catalunya, the National Science Foundation, nor the names of any of the copyright holders of PUPIL may be used to endorse or promote any products derived from this Software without specific, prior, written permission from at least one of the three Original Design and Continued Evolution contributors listed below.

II. Acknowledgments

Partial support from U.S. National Science Foundation ITR Grant DMR-0325553 is acknowledged with thanks. This material is based upon work also supported by the National Science Foundation under the following programs: Partnerships for Advanced Computational Infrastructure, Distributed Terascale Facility (DTF) and Terascale Extensions: Enhancements to the Extensible Terascale Facility. The authors also acknowledge the University of Florida High-Performance Computing Center and Teragrid (Grants TG-MCA05S010 and TG-CHE060072T) for providing computational resources and support.

III. Trademarks

“Gaussian 03” is a registered trademark of Gaussian, Inc. (340 Quinnipiac St Bldg 40, Wallingford, CT 06492, USA).

We have endeavored to be scrupulous regarding trademarks. If we have overlooked a trademark reference, we will be pleased to correct the oversight. Please contact one of the three “Original Design” contributors listed below.

IV. Contributions

- *Original Design, MNDO97 interface:*
  - Juan Torras — EUETII, Univ. Politècnica de Catalunya, Spain
  - Erik Deumens — QTP, University of Florida, USA
  - Samuel B. Trickey — QTP, University of Florida, USA

- *Amber and Gaussian Interfaces*
  - Adrian Roitberg QTP, University of Florida, USA
  - Gustavo M. Seabra QTP, University of Florida, USA
  - Benjamin Roberts QTP, University of Florida, USA
- *DL_POLY and SIESTA Interfaces*
  - Hai-Ping Cheng QTP, University of Florida, USA
  - Chao Cao QTP, University of Florida, USA
  - Yao He Yunnan University, China
- *Domain Identifier*
  - Krishna Muralidharan Matl. Sci. Eng., University of Arizona, USA
- *Continued Evolution, User Guide:*
  - Juan Torras EUETII, Univ. Politècnica de Catalunya, Spain
  - Erik Deumens QTP, University of Florida, USA
  - Samuel B. Trickey QTP, University of Florida, USA
  - Gustavo M. Seabra QTP, University of Florida, USA
  - Benjamin Roberts QTP, University of Florida, USA

V. Standard Citations

Scientific papers and presentations incorporating results obtained using PUPIL must reference the code as follows:

> "PUPIL, **P**rogram for **U**ser **P**ackage **I**nterfacing and **L**inking, a software product of the University of Florida Quantum Theory Project, J. Torras-Costa, E. Deumens, S.B.Trickey, H-P.Cheng, C.Cao, Y. He, K. Muralidharan, A. Roitberg, G. M. Seabra and B. P. Roberts."

Users are also requested to cite at least one of the following three papers about PUPIL:

[1] *"PUPIL: A systematic approach to software integration in multi-scale simulations"*, J. Torras, Y. He, C. Cao, K. Muralidharan, E. Deumens, H.-P. Cheng, and S. B. Trickey. Computer Physics Communications **177**, 265-279 (2007).

[2] *"Software integration in multi-scale simulations: the PUPIL system"*, J. Torras, E. Deumens, and S. B. Trickey. J. Computer-Aided Materials Design, **13**, 201-212 (2006).

[3] *"A versatile AMBER-Gaussian QM/MM interface through PUPIL"*, J. Torras, G. M. Seabra, E. Deumens, S. B. Trickey, and A. E. Roitberg. J. Comput. Chem. **29** 1564-1573 (2008).

VI. Table of Contents

1. INTRODUCTION 6

2. INSTALLATION 8

2.1 PREREQUISITES 8
2.2 PUPIL DIRECTORY STRUCTURE 8
2.3 BUILDING THE PLATFORM-INDEPENDENT COMPONENTS 10
2.4 BUILDING THE PLATFORM-DEPENDENT COMPONENTS 10
2.4.1 BUILDING LOOSELY COUPLED USER PACKAGES. 10
2.4.2 BUILDING TIGHTLY COUPLED USER PACKAGES 11
2.4.2.1 Conditioning Source Code 11
2.4.2.2 Compiling PUPIL Libraries and Binaries 12
2.4.2.3 Linking User Package Objects with PUPIL Libraries 13
2.5 TESTING YOUR PUPIL INSTALLATION 14

3. RUNNING SIMULATIONS 16

3.1 PREPARING SIMULATION INPUT FILES 16
3.2 THE RUN SHELL SCRIPT 16
3.3 OUTPUT SIMULATION FILES 17
3.3.1 MANAGER OUTPUT FILES 17
3.3.1.1 AppServer.log 17
3.3.1.2 output.xml 18
3.3.2 WORKER OUTPUT FILES 18

4. GUI – GRAPHICAL USER INTERFACE 19

4.1 SIMULATION 19
4.1.1 NEW/MODIFY SIMULATION. 19
4.1.2 CALCULATION UNITS SPECIFICATION. 20
4.1.2.1 Force Generation (QM) 21
4.1.2.2 Embedding Rules 23
4.1.2.3 QM Applications Currently Implemented 25
4.1.2.4 Molecular Dynamics (MD). 27
4.1.2.5 MD Applications Currently Implemented 27
4.1.2.6 Domain Identification (DI) 27
4.1.3 KEYMM/KEYQM MAPPING 28
4.1.4 SHOW SIMULATION TREE 29
4.2 RESULTS 29
4.2.1 QM SIMULATION SUMMARY 30
4.2.2 EXTRACT XMOL FILE 30

5. XML SIMULATION FILE 31

5.1 THE SIMULATIONROOT ELEMENT 31
5.1.1 THE ATOMDICTIONARY ELEMENT 31
5.1.2 THE RESIDUEDICTIONARY ELEMENT 31
5.1.3 THE KEYMM ELEMENT 31
5.2 THE SIMULATION ELEMENT 31
5.3 EXAMPLE DATA.XML FILE 32

1. INTRODUCTION

PUPIL, **P**rogram for **U**ser **P**ackage **I**nterfacing and **L**inking, is a software environment – the program – to allow developers to accomplish an increasingly important task, namely, the quick, efficient linking of several independent pieces of software – the “user packages” – that have been and are actively being developed by researchers. PUPIL is general and can be used to link user packages from any scientific or engineering domain. However, it was originally developed with multi-scale simulation in materials physics and chemistry in mind, and several of the interfaces included in this release show that heritage.

This manual explains how to download, build, and install PUPIL. It also explains how to set up a computation and perform a calculation. In this Introduction, we give a brief overview of PUPIL’s architecture so you can gain a basic understanding of how it works. However, to get a thorough understanding, you should read the publications listed in the Standard Citations section above. Also, please keep in mind the Standard Citations requirement listed in the preliminary material of this manual.

The design philosophy of PUPIL is to provide an environment for the software developer of user packages to do simulations in which data and simulation control are transferred from one user package to another in a straightforward manner. A design requirement is to do this without creating a monolithic, single-threaded code. A further design requirement is that any changes in any of the user packages should be small and systematic. To make such changes, one obviously must understand the user package, but a PUPIL design objective is to avoid the need to have a complete and exhaustive understanding such as usually is required when one wants to create a combined user package from multiple, independently developed user packages.

PUPIL itself acts as a **supervisor** program, coordinating execution and communication between the user packages, each of which acts as a **calculation unit** (CU). The supervisor is implemented as a distributed program with one **manager** and several **workers**, one worker for each CU. The manager and the workers communicate using the CORBA[1] (Common Object Request Broker Architecture) protocol. Workers communicate with tightly coupled user packages via subroutine calls, and with loosely coupled user packages through data files. The manager and the worker codes are written in Java. The worker’s Java code calls C-code in the CU through the JNI[2]Java Native Interface). The CUs, often written in Fortran or C, communicate with their workers via a PUPIL library, written in C.

Throughout this manual, we observe a few typographical conventions. Commands to be typed in at a prompt are given in `monospace` and follow a `$` sign (which denotes the shell prompt). Directory and file names, where these are not given as part of a command, are shown in *italics*.

---

1 Object Management Group, OMG Common Object Request Broker Architecture: Core Specification. Web site: http://www.omg.org/

2 Sun Microsystems, Inc. Java Native Interface 5.0 specification

Throughout, the terms “coordinates” and “system coordinates” mean coordinates of nuclei and/or residues. (Electronic coordinates are internal to Quantum CUs.)

2. INSTALLATION

This chapter discusses in detail the steps needed to build and install PUPIL on your computer system. This information is most valuable to the system administrator. In addition, programmers who want to interface their user package with other user packages through PUPIL will find the information essential. Researchers who want to use PUPIL together with a set of already configured user packages do not have to read this chapter.

2.1 Prerequisites

To build and install PUPIL, the following software components must be installed on your computer system:

- SDK 1.5 or later — http://java.sun.com/javase/index.jsp
- Apache Ant — http://ant.apache.org
- GNU make ("gmake") — http://www.gnu.org/software/make

The present release of PUPIL includes support for five user packages. To use any one of them, you must have access to it or else obtain a licensed copy and install the package on your computer system. We strongly recommend testing each user package by itself before using it with PUPIL.

- User Packages:
  - AMBER v10 — http://ambermd.org/
  - DL_POLY v2.18 — http://www.cse.clrc.ac.uk/msi/software/DL_POLY
  - Gaussian 03 — http://www.gaussian.com/
  - Siesta v2.0 — http://www.icmab.es/siesta/
  - MNDO97 (Walter Thiel, v 5.0 April 1998)

To view molecular and material structures in its graphical user interface, PUPIL uses the Jmol tool, which is included in the PUPIL library as an external jar library.

- Jmol: — http://jmol.sourceforge.net

2.2 PUPIL Directory Structure

Download the latest release of PUPIL from http://pupil.sourceforge.net. Extract all files from the .tar file. In the resulting directory, you will find the following subdirectories:

- *bin/* — Compiled binaries
- *build/* — Working directory used by Ant
- *doc/* — PUPIL manual and javadoc of class diagram
- *lib/* — PUPIL library
- *run/* — Directory containing the shell scripts needed to run PUPIL
- *src/* — Source code
- *tests/* — Directory containing tests of the PUPIL system and some binaries

Inside these directories, you will find the following files:

- *src/* directory:
  - *build.xml* — Apache Ant XML build file (the equivalent of a Makefile), with instructions for building the Java application.
  - *pupil.idl* — IDL file for a Java Sun SDK 1.4 corba compiler
  - *cInt/* — This directory contains the PUPIL machine-dependent code, including source code and shell scripts for the C interface (JNI), source code for interfaces between PUPIL's Java programs and CUs written in Fortran, and patches for CUs themselves.
  - *jmol/* — This directory contains the code for Jmol, modified to work with the PUPIL GUI.
  - *PUPIL/* — This directory contains the Java source code for the PUPIL manager and workers.
- *tests/* directory:
  - *scripts/* — Main scripts to run PUPIL tests.
  - *stubs/* — Tests using stub programs: stubMD, stubQM and stubDI. Some tests use only stub programs, while others also use installed QM CUs (MNDO, Siesta, Gaussian, *etc*.)
  - *sio2/* — Simple tests for the DL_POLY binary with the installed QM CUs.
  - *ala-di/* — Simple test of alanine dipeptide QM/MD in explicit water, involving the Amber binary with both the stubQM and the Gaussian 03 binaries. (If Gaussian03 is not to be used, the test makefile should be edited accordingly.)
  - *ala3/* — Simple link atom test for the system made of a simple peptide, ACE-$(ALA)_3$-NME, in explicit water. This test uses the AMBER and Gaussian 03 packages.
- *run/* directory:
  - *run.sh* — A template shell script to execute a PUPIL calculation. This shell script should be modified for any particular calculation before execution.
  - *pupil_paraRun.sh* — A template shell script to use an MPI-compatible QM program with PUPIL.
  - *clean.sh* — A shell script to remove output and temporary files from a PUPIL execution directory. Since PUPIL

uses standard names for its output, this script should in most cases be usable as is.

Throughout this manual, we will assume that the environment variable **PUPIL_PATH** contains the root directory of the PUPIL distribution. This variable can be set in shells such as **sh** or **bash** by the following command, which can also be added to your *.bashrc* file:

```
$ export PUPIL_PATH=/path/to/pupil
```

where `/path/to/pupil` is replaced by the directory in which PUPIL is installed (such as */usr/local/pupil-1.2.1* or */opt/pupil-1.2.1*).

2.3 Building the platform-independent components

The graphical user interface (GUI) (see below) and PUPIL Supervisor are implemented in Java. Compilation is done using `ant`.

1. Go to the PUPIL src directory:

```
$ cd $PUPIL_PATH/src
```

2. Compile PUPIL using ANT:

```
$ ant clean
$ ant
```

Two Java jar files will be created and stored in the `$PUPIL_PATH`*/lib* directory:

- *PupilServer.jar* — The PUPIL supervisor code and the shared PUPIL worker codes to be used in any simulation. See Chapter 3 for execution details.
- *PupilClient.jar* — The graphical user interface. See Chapter 4 for execution details.

A JavaDoc describing the PUPIL conceptual model (data structures and their relationships) will also be created in *$PUPIL_PATH/doc/pupilDoc/index.html*.

2.4 Building the platform-dependent components

The procedure to build the platform-dependent PUPIL binaries and libraries is described in this section. The steps are listed in the order in which they should be executed to compile the system.

2.4.1 Building loosely coupled User Packages.

Some QM User Packages run as independent executables called by the PUPIL system. These packages do not require any source-code modification or recompilation to work with PUPIL. This mode of CU operation is called Start-Stop (SS), because a new instance of the QM CU is executed at each force evaluation. Loosely coupled CUs should be compiled by themselves as usual, following their own instructions. The resulting binaries should then be stored in the *$PUPIL_PATH/bin* directory as:

{binary}{Operating System}

For instance, for the MNDO 97 User Package on a system running Linux, the mndo binary should be stored as *$PUPIL_PATH/bin/mndoLinux*. Alternatively, a symbolic link may be created in the *$PUPIL_PATH/bin* directory, pointing to the executable:

```
$ cd $PUPIL_PATH/bin
$ ln –s <path_to_executable> ./{executable_name}{Operating_System}
```

Currently, the User Packages that run in Start-Stop mode with PUPIL are:

– MNDO 97 ==> *mndo{Operating System}*, e.g., *mndoLinux*
– SIESTA v2.0 ==> *siesta{Operating System}*, e.g., *siestaLinux*
– Gaussian 03 ==> *g03{Operating System}*, e.g., *g03Linux*

Some User Packages require extra data files, such as basis set libraries. These files must be visible at the location specified when the User Package executable was built, on every computer where the package will be run by PUPIL. These data files do not need to reside inside the *$PUPIL_PATH* tree.

2.4.2 Building Tightly Coupled User Packages

These User Packages require source-code modification and linking against PUPIL's libraries to be used with the PUPIL interface. Three steps are required to build such tightly coupled User Packages: Conditioning the source code (see next Subsection), compiling PUPIL libraries, and finally, building the new tightly coupled User Package binaries linked to the new PUPIL libraries obtained in the previous step.

2.4.2.1 Conditioning Source Code

The User Packages that have an interface to interact directly with the PUPIL system should be conditioned (i.e., modified) before their compilation. The exact conditioning process varies among User Packages.

– *DL_POLY and SIESTA*

  Source code patches for these programs are distributed with PUPIL. The steps to patch the original source code are the following:

  1. Create a new directory inside your file system, with all the source code for the User Package to build the new Calculation Unit (DL_POLY or SIESTA) to plug into PUPIL. This source code will be modified during the compilation.
  2. If not already downloaded, download the patch file from http://pupil.sourceforge.net.
  3. Go to the directory containing system-dependent PUPIL files:
     ```
     $ cd $PUPIL_PATH/src/cInt
     ```
  4. Extract the contents of the patch file within this directory. A new *patches/* directory will be created.
  5. Enter this *patches/* directory:
     ```
     $ cd patches
     ```

6. Edit the patching shells **.sh* and put the correct directories in the configuration section of the script. These directories correspond to the place where the source code of the User Package *to be patched* with the PUPIL Interface is located. (See the README file)
7. Execute the patch shell.

The patches currently tested with PUPIL package correspond to DL_POLY v2.18 and SIESTA v2.0.

– *AMBER 10*

The current Amber 10 source code already includes the necessary modifications to interface with the PUPIL library, so further conditioning is not necessary to support use with PUPIL. See Section 2.4.2.3 for instructions on compiling and linking the "sander" binary.

2.4.2.2 Compiling PUPIL Libraries and Binaries

1. First set the environment variable PUPIL_PATH with the path where the PUPIL distribution has been stored. For example, in **sh** or **bash**,

```
$ export PUPIL_PATH=/path/to/pupil
```

where */path/to/pupil* is replaced by the path to the PUPIL directory on your system.

2. Add the PUPIL library path to the $LD_LIBRARY_PATH environment variable:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH\:$PUPIL_PATH/lib
```

Note: On some platforms it may be necessary to specify the Java libraries in the LD_LIBRARY_PATH as well, for example:

```
$ ex-
port LD_LIBRARY_PATH$JAVA_HOME/jre/lib:$JAVA_HOME/jre/lib/$PROC:$JAVA
_HOME/jre/lib/$PROC/server:$LD_LIBRARY_PATH
```

where $JAVA_HOME is a variable with the full path to the SUN Java installation (or equivalent) and $PROC is the type of processor in your computer. *Check your Java installation for specific details.*

3. Configure the build directories for your system and build the Interface library and new CU binaries with the following steps

   a. Change to the *cInt/* directory:

   ```
   $ cd $PUPIL_PATH/src/cInt
   ```

   b. Edit the *configure.sh* file and put in the correct directory paths for the User Packages that you are going to compile (the tightly coupled CUs that need to be patched), in order to customize the installation. Because of the platform dependence for the compilation and Java libraries location, the user has to edit and modify the Java libraries path depending on the kind of processor in the user's platform. Similarly, the user may also have to modify the compilation and linking parameters for the user's specific case if the default values are not appropriate to the current compilation platform.

c. Execute, in order, the following commands:

```
$ ./configure.sh
$ gmake
$ gmake install
```

There are three PUPIL libraries that the *Makefile* will store in the *$PUPIL_PATH/lib/* directory:

- *libPUPIL.so* — Main Interface between User Packages and the Java PUPIL supervisor.
- *libPUPILTime.so* — Library with routines to compute timing.
- *libPUPILBlind.so* — Library with stub functions. If a particular User Package does not define certain functions (usually because it does not need them), the stub library will provide a dummy replacement, so that the linking step will succeed without error.

  Note: When linking any program to PUPIL, libPUPILBlind.so always should be placed at the end of the link command line.

By default, three binaries will be made as stubs to test the functionality of the manager in the simulation. Each one corresponds to the three roles that any application can play in a multi-scale simulation. All of them will be installed in the `$PUPIL_PATH/bin` directory. The three binaries are:

- *stubMD{OperatingSystem}* — Simulates the calls to/from the PUPIL library by a Molecular Dynamics Calculation Unit for a few steps.
- *stubDI{OperatingSystem}* — Simulates the calls to/from the PUPIL library by a Domain Identification Unit.
- *stubQM {OperatingSystem}* — Simulates the calls to/from the PUPIL library by a Quantum Mechanics Calculation Unit.

2.4.2.3 Linking User Package Objects with PUPIL Libraries

All User Packages (CUs) tightly coupled with PUPIL must be linked with the PUPIL libraries previously compiled and the platform-dependent Java libraries (see Section 2.4.2.2). The environment variables LD_LIBRARY_PATH, PUPIL_PATH, and JAVA_HOME also must be set.

– *DL_POLY and SIESTA*

1. Change to the *cInt/* directory:

   ```
   $ cd $PUPIL_PATH/src/cInt
   ```

2. Edit *configure.sh* and add the path to the directory where the code to be patched (DLPOLYCODE and SIESTACODE variables) is stored.

3. The patched source code contains a *Makefile* already prepared for a default machine but a sanity check is strongly recommended. Follow instructions in subsection 2.4.2.2. Make sure you have the correct *arch.make* file for SIESTA code compilation.

4. Execute, in order, the following commands:

```
$ gmake uninstall
$ gmake clean
$ ./configure.sh
$ gmake
$ gmake install
```

– *AMBER 10*

Before you continue, make sure you have applied all the bugfixes for Amber (available from http://ambermd.org/bugfixes), and that you can build and successfully test a fully functioning stand-alone (serial) version of Amber from the patched code. See the Amber manual for details on Amber installation and testing.

1. Change to the *$AMBERHOME/src/sander/* directory:

```
$ cd $AMBERHOME/src/sander
```

2. Compile the modified sander code and link with PUPIL libraries:

```
$ make sander.PUPIL
```

3. Copy the PUPIL-coupled *sander.PUPIL* binary to the *$PUPIL_PATH/bin* directory, following the PUPIL naming convention:

```
$ cp sander.PUPIL $PUPIL_PATH/bin/sander{Operating System}
```

Alternatively, you may create a symbolic link:

```
$ cd $PUPIL_PATH/bin
$ ln -s $AMBERHOME/src/sander/sander.PUPIL sander{OS}
```

2.5 Testing your PUPIL installation

To test the compiled PUPIL libraries, CU binaries, and the Java supervisor, execute the *Makefile* located in *$PUPIL_PATH/tests*. All the binaries, or symbolic links to them, must be present in the *$PUPIL_PATH/bin* directory following the naming convention used above. This *Makefile* tests a simple SiO_2 molecule using all the possible combinations of the stub, MNDO, SIESTA and Gaussian 03 as QM Calculation Units and stub and DL POLY as the MD Calculation Units. It assumes that *mndo*, *dlpoly*, *siesta, g03* and all the *stubs* are stored in the *$PUPIL_PATH/bin* directory. Any deviation from this convention must be accounted for directly in the *Makefile* in the *$PUPIL_PATH/tests* directory.

1. Change to the tests directory

```
$ cd $PUPIL/tests
```

2. Execute the tests:

```
$ gmake clean
$ gmake
```

This command tests the whole system.

3. Check the results:

```
$ gmake check
```

When all the simulations have completed, this last command will extract certain information from the *AppServer.log* file and compare the extracted text with the contents of a previously stored file to decide whether the test was passed or not.

For more details see the *Makefile* in the $*PUPIL/tests* directory.

3. *RUNNING SIMULATIONS*

3.1 Preparing simulation input files

The PUPIL system allows the user to link all the User Packages (CUs) easily, but it is the user's responsibility to prepare all the input files for each CU so that each one will work correctly within the simulation. The PUPIL manager *does not check* the validity of the input files for the individual CUs. Please refer to the respective User Package manuals for instructions on input file preparation.

The coordinates of the molecular system, cluster, or extended system, as well as the classical type of each atom, are specified in the Molecular Dynamics Unit input files. Different potentials for the same element can be represented as different atom types. For example, in silica with water simulations we could have two kinds of classical Oxygen atom, Oxygen from silica and Oxygen from water.

The input file for the Quantum CU supplies the general quantum variables except for the system coordinates. The quantum types of the particles also are obtained from this input file. The PUPIL system parses this file to record the different quantum particles that the user has defined in the multi-scale simulation. Following the same example of silica and water, we could assign different basis sets to the Oxygens belonging to the silica and to those belonging to water.

With the two kinds of particles, classical and quantum, recorded, PUPIL creates a default mapping between them, indexed by the atomic number. The default mapping can be modified using the PUPIL GUI (see section 4.1.3).

3.2 The run shell script

The *run.sh* script included in this package has several sections that are described below in the order that they appear in the script:

– *PBS directives*

  PUPIL is designed to work with a cluster queue system. This section has PBS directives to submit the job using a PBS queue manager. The user should modify this part to correspond with local usage requirements and practices.

– *Configuration variables*

  In this section, the user should specify the working directory path (BASEDIR environment variable) and the location of the *data.xml* simulation file (DATA environment variable, which can vary for each simulation job). Note: At present, the simulation file *must* be named *data.xml*.

– *Initializing variables*

  Based on environment variables and the "configuration variables" set before, the script initializes a number of internal variables.

– *Execution of CORBA Name Server*

  To exchange information between the different CUs working at the same time in the simulation, we start a CORBA Name Server from the Java Sun Package. The script will try to start the nameserver on an unused port on the machine that runs the script, starting at port 3000 and trying up to port 3010. If no unused port can be found within this range, PUPIL will fail.

– *Execution of Application Server*

  The PUPIL supervisor is started and it runs the desired CUs.

– *Stopping Services*

  When the simulation is finished or stopped, all the services, including the CORBA name server, must be stopped properly. Sometimes, even if the PUPIL system is shut down correctly, the system semaphores used to synchronize the applications are not yet deleted. This section deletes all unused system semaphores that remain after finishing the PUPIL application.

3.3 Output simulation files

3.3.1 Manager output files

There are two types of files that consolidate all the outputs from PUPIL simulations, AppServer.log and the output files from distributed CUs.

3.3.1.1 AppServer.log

This file is written by the PUPIL Manager. All the CUs exchange information and events with the Manager, which is in charge of writing them in this log file. There are four different levels of output detail:

-1	Only errors are printed.
0	Normal comments and errors are printed (default level).
5	Debugging information, except for system coordinates, is printed.
10	All coordinates and debugging information are printed.

Every entry in the log file has the origin of the message between << >> at the beginning of the line. The most common sources of comment entries are the following:

– *CoordinatesServer*

  This is the general MD worker, which receives the classical system coordinates and generates the QM system coordinates.

– *CoordIntfc*

  This Java class implements the CORBA server for the CoordinatesServer. Usually, CoordIntfc receives the quantum forces from the general QM worker and the quantum zone from the General DI worker.

– *ForcesServer*

  This is the general CycleQM worker, which receives the quantum forces from the quantum packages through the PUPIL library and sends those forces to CoordIntfc.

– *ForcesIntfc*

  This Java class implements the CORBA server for ForcesServer. It receives the quantum system coordinates and puts them into the cycleQM package through the PUPIL library.

– *DomainsServer*

  This is the general DI worker, which receives the atom numbers that belong to the quantum zone from the DI packages through the PUPIL library and sends that information to CoordIntfc.

– *DomainsIntfc*

  This Java class implements the CORBA server for DomainsIntfc. It receives the classical particle coordinates, atom types, and other variables to pass through the PUPIL library to the program that will determine the quantum domain.

– *PUPIL.Domain*

  These specialized Java classes from the System Domain are responsible for any specific CU behavior, such as THMNDOQMJob, SiestaQMJob, etc. See References [1] and [2] listed in the Standard Citations.

3.3.1.2 output.xml

This file has a structure similar to that of *data.xml* (see Chapter 5) but with all the intermediate results obtained in the simulation. The user decides how many steps will be taken before writing a new record to *output.xml*. This output file is useful for following the multi-scale simulation. However, the output can become quite large when the physical system has a large number of particles. If the number of particles is very large, writing this file may exceed available memory, causing PUPIL to crash. The memory resources are monitored in the *AppServer.log* file. To avoid crashing PUPIL when the physical system has a very large number of particles, the user may have to consult whatever intermediate files the CU may provide to analyze the simulation results instead of adding new intermediates steps to be stored in the *output.xml* file (see Section 4.1.2.4).

3.3.2 Worker output files

The standard output and standard error channels for all general workers are redirected to files (one standard output file and one standard error file for each worker). All the normal output from the workers is contained within these files. Errors that occurred in any worker can be monitored in these files as well as in the general log file. The debug messages from the worker–PUPIL interfaces and the PUPIL C libraries may also be found in these files.

4. GUI – Graphical User Interface

PUPIL's graphical user interface helps the user to build the input file (*data.xml*) for a multi-scale simulation. This chapter explains the GUI options.

The GUI is started using the following command:

```
$ java -jar $PUPIL_PATH/lib/PupilClient.jar
```

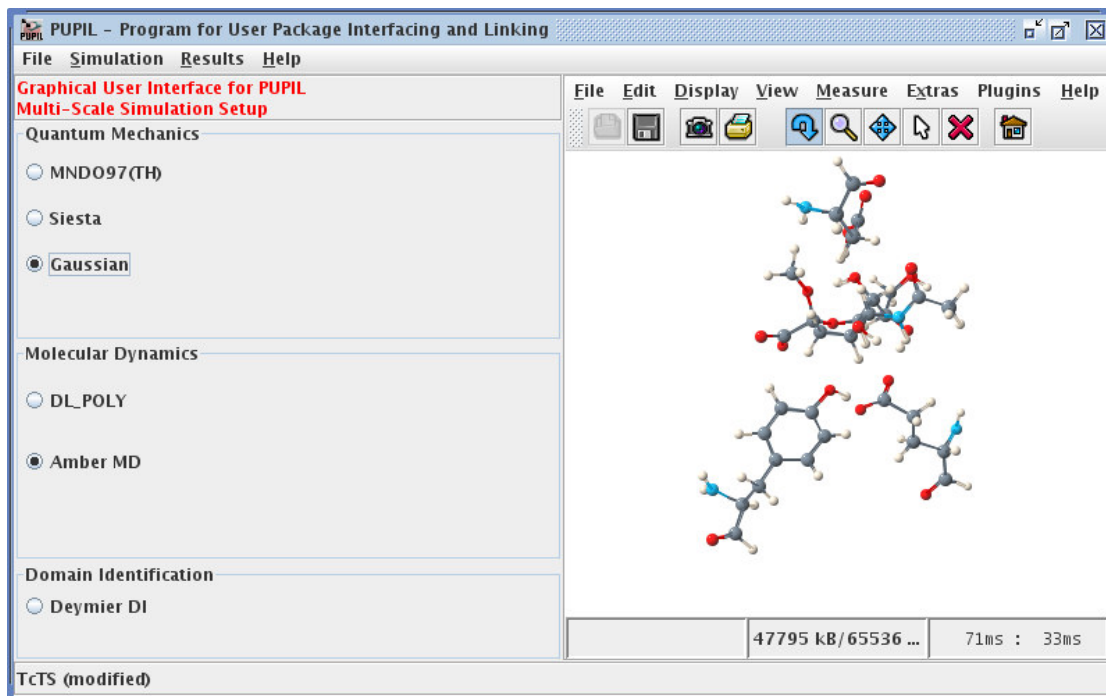


Figure 1. PUPIL GUI showing the state immediately after loading the QM Calculation Unit files. In this example, Gaussian03 is being used as the QM CU.

4.1 Simulation

The simulation input consists of a brief description of the whole simulation task and the necessary information to run each Calculation Unit. All this information will be stored in memory in the same way as in the *data.xml* input file created to run the simulation (see Chapter 5).

4.1.1 New/Modify Simulation.

In the first step, a new simulation must be created via the *Simulation* → *New Simulation Input* menu option. The following fields must be completed:

– *Simulation Name*

  A user-defined name for this simulation.

– *Base Directory*

This is the path of the working directory in the file system where PUPIL should expect to find input, and where the output and temporary files will be created. A period (decimal point or full stop, ".") may be used to indicate the directory in which the initial shell script (*run.sh*) is executed.

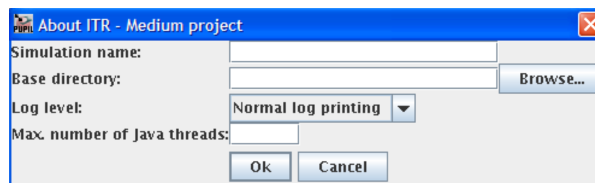


Figure 2. New simulation dialogue box

– *Log Level*

Shows the level of printing from the PUPIL system during the simulation. Three possibilities are accessible from the GUI:

– Without log printing: Only errors will be printed.

– Normal log printing: Basic output will be printed, allowing the user to follow the simulation's progress.

– Debug log printing: Detailed information about the data (except for particle coordinates) from each worker will be printed at each simulation step.

Though not accessible from the GUI there is a fourth, extremely verbose log level which prints the coordinates of each atom at every step. This level may be accessed by giving the optPrint element in the input XML file (*data.xml*; see Chapter 5 for a brief description of this file) a value of 10.

– *Max. number of Java Threads*

For shared-memory (SMP) machines, this input allows one to specify the number of threads to be created by the Java code, and with which the PUPIL Manager will work in parallel. Use of Java threads improves the speed of the quantum zone – point charge correction significantly. If performing such a correction, we recommend requesting as many threads as the number of processors permitted to PUPIL on your computer (or the head node if you are using MPI and several machines).

Version Note*: At present, only the forces-over-point-charge (QZ-PC correction for PC) calculation takes advantage of this facility. In the future, Java threads will be extended to the whole PUPIL core.*

4.1.2 Calculation Units specification.

The main GUI panel (Figure 1) is divided in two sections. On the left is a list of the CUs that are supported by the PUPIL system.

> ***Note****: CUs for which support is under development may show in the panel but not be available at your site. Also, CUs will be listed even if not installed at your site.*

On the right is the main window for the Jmol application (http://jmol.sourceforge.net) that helps visualize the classical and quantum system read by PUPIL from the input files discussed in section 3.3. JMol has been embedded into the PUPIL GUI.

The CU panel is divided into three sections, one for each role that a CU can play in the PUPIL multi-scale simulation.

You must specify one *Molecular Dynamics* and one *Force Generation* (QM) method for the simulation. Use of a *Domain Identification* CU is optional. It can be avoided by specifying a quantum zone using a set of rules that are associated with each calculation method. All CUs have a common set of parameters to be assigned when selected:

– *Executable.*

  The user specifies the path and name of the binary that will be associated with the CU. A copy of that binary will be placed in a subdirectory within the simulation's working directory; this copy will be executed during the simulation.

– *Num. processors.*

  Warning! *This option works only with QM applications that have been compiled to work within the MPI environment. If your QM program does not support MPI, or you wish to run it in serial mode, you must enter "0" here. Entering any other value, including 1, will cause PUPIL to fail.*

  This is the number of MPI processors associated with the CU. A value of "0" corresponds to execution on one processor for the whole simulation. A shell script template, *pupil_paraRun.sh* (from the PUPIL software), is used to build a proper script to start the QM MPI calculation. The template is found in the *$PUPIL_PATH/run/* directory. At each execution, PUPIL enters appropriate values for all the internal environment variables contained in the template. Prior to this, however, the user must:

  1. Edit the template (*pupil_paraRun.sh*) to match the MPI package installed on the target system, and
  2. Copy the result into the simulation directory where the PUPIL starting run shell script (*run.sh*) has been placed.

4.1.2.1 Force Generation (QM)

To obtain the quantum forces, some common parameters must be specified for each CU involved in the simulation.

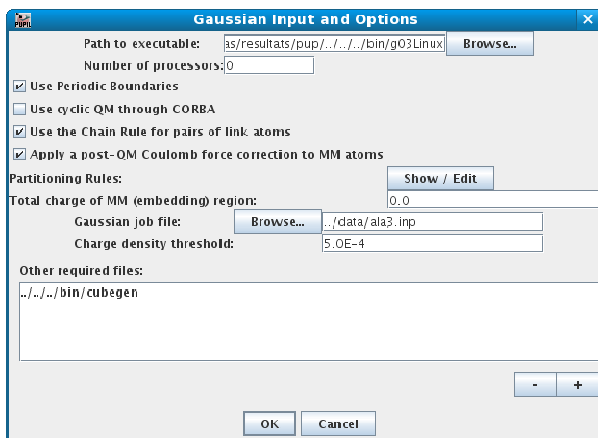


Figure 3. Gaussian QM specification

– *Use Periodic Boundaries*

  This option is needed only if the classical system has 3D periodicity in a parallelepiped or cubic MD cell. Only orthogonal unit cell vectors are allowed so far. (No hexagonal unit cells). If this option is selected, PUPIL will translate the atomic coordinates using

the periodicity of the system, such that the quantum zone ends up at the center of the unit cell for the QM calculation.

Version Note*: Full periodic boundary conditions are not implemented in the present release of PUPIL.*

- *Use cyclic QM (through CORBA)*

  Warning! *This option applies only to tightly coupled QM packages. If this option is checked and the binary does not have the corresponding cycleQM behavior compiled with a proper PUPIL interface, the simulation will fail in a deadlock.*

  This option tells the PUPIL Manager that the current CU will have the CycleQM behavior. The binary will be put into execution once, then used repeatedly. It will be restarted only when the quantum zone changes.

- *Chain Rule for pairs of link atoms*

  If this option is checked *and the simulation has link atoms in the embedding zone* (see *Embedding Rules*, below), the Chain Rule will be applied to every pair of link atoms to distribute the force over the link associated with them.

- *QZ-PC correction for PC*

  Most QM user packages do not compute the force exerted by the QM region on classical atoms (which typically are represented to the QM program as point charges). If this option is checked and the simulation has point charges in the embedding zone (see *Embedding Rules*), a simple integration between the point charges and the electronic density is done over the system volume (See reference [3]). This approximation attempts to calculate and correct this force component, modifying the forces on the classical atoms associated with the point charges.

- *Charge density threshold*

  A Gaussian 03 Cube file is used to obtain the QM electronic density at each MD/MM step. The Cube file, produced using the Gaussian 03 utility *cubegen*, is a text file containing an approximate description of the charge distribution of the QM region. It uses a regularly spaced grid , each with a partial charge. Those grid-point charges are, in turn, used to calculate the electrostatic force exerted by the QM region on MM atoms. It is common for many of the grid-point charges to be negligible, and calculation of the electrostatic force is faster if these points are skipped. The charge density threshold is the charge level below which a grid point will be skipped. A typical value for the charge density threshold is 1×10^{-6}.

- *Total charge of embedding region*

  Introduction of a link atom in the quantum zone can cause a charge neutrality violation. To correct this, PUPIL will adjust the total charge of the embedding region to the current value introduced by the user.

  Note*: the charge introduced in this box is the net charge of the embedding (classical) region only.*

Editing embedding rules

Direct ATOM types assignment rules Add Delete

Zone	Unit Number	New Unit type
EMB1	899	PC.LEU.H
EMB1	900	PC.LEU.CA
EMB1	901	PC.LEU.HA
EMB1	902	PC.LEU.CB
EMB1	903	PC.LEU.HB2
EMB1	904	PC.LEU.HB3
EMB1	905	PC.LEU.CG
EMB1	906	PC.LEU.HG
EMB1	907	PC.LEU.CD1
EMB1	908	PC.LEU.HD11

Neighbouring ATOM types assignment rules Add Delete

Zone	Ini unit type	rMin	rMax	End unit type	New Distan...
EMB1	PHE.C	0.0	1.8	H	1.0
EMB1	ASN.N	0.0	1.8	H	1.0
EMB1	SER.C	0.0	1.8	H	1.0
EMB1	PRO.N	0.0	1.8	H	1.0
EMB1	ALA.C	0.0	1.8	H	1.0
EMB1	SER.N	0.0	1.8	H	1.0

Direct RESIDUE types assignment rules Add Delete

Zone	Unit Number	New Unit type
QZONE	230	Default
QZONE	342	Default
QZONE	372	Default
QZONE	373	Default
QZONE	59	Default

Neighbouring RESIDUE types assignment rules Add Delete

Zone	Ini unit type	rMin	rMax	End unit type	New Dista...
EMB2	ALA	0.0	100.0	Default	NO
EMB2	ARG	0.0	100.0	Default	NO
EMB2	ASH	0.0	100.0	Default	NO
EMB2	ASN	0.0	100.0	Default	NO
EMB2	ASP	0.0	100.0	Default	NO
EMB2	CYS	0.0	100.0	Default	NO
EMB2	GLN	0.0	100.0	Default	NO
EMB2	GLU	0.0	100.0	Default	NO
EMB2	GLY	0.0	100.0	Default	NO
EMB2	HIE	0.0	100.0	Default	NO

OK Cancel

Figure 4. The dialog box for specifying the embedding rules and assigning atoms to classical or quantum zones.

– *Other required files*

This box allows the user to specify other required programs and files. Particularly, the program *cubegen* can be added here using the "+" button. *Cubegen* is a utility which is shipped with Gaussian 03 and which produces a grid of point charges approximately representing the charge density of a QM region. Currently, if Gaussian 03 is used as the QM calculation unit, *cubegen* is needed to perform QZ-PC corrections (see Section 4.1.2.3).

4.1.2.2 Embedding Rules

Although this dialogue box is contained in the definition of the QM CU, its importance merits a separate discussion.

When no embedding rules are specified and no DI program is running, the entire classical system will be considered as a single quantum zone and will be mapped following the default mapping rules between the classical and quantum kinds of atoms (see section 4.1.3), based on atomic numbers.

In this dialogue (Figure 4), the user has the option to assign, with a specific set of rules, a fixed quantum zone and its embedding particles or just the embedding particles around a given quantum zone obtained from a DI Calculation Unit. To assign those rules, there is a specific panel in the PUPIL GUI that allows the user to apply four kinds of basic rules to atoms and/or residues:

– *Direct atom type assignment*

This region is used to define a relationship between a classical particle (identified by its atom number) and the particle associated with it in the quantum calculation (i.e., a point charge or a full QM atom), *on a particle-by-particle basis*. The user must specify here all particles that will not be accounted for correctly by any of

the more general rules. This definition may assign the atom as a quantum atom or as a member of the first embedding layer or the second embedding layer (but no farther). Also, the particle must be given a quantum-atom type or specified as a point charge.

Residues with both quantum and classical parts cannot be fully assigned to either the QM zone ("QZONE") or either of the two classical zones ("EMB1", "EMB2") defined below. As a result, all atoms in these residues, except the classical counterparts of possible *link atoms*, must be specified individually here, and assigned to the corresponding point charge or QM atom.

– *Neighboring atom type assignment.*

This rule has a different philosophy from the direct atom type assignment rule. The neighbor rule will assign *all* particles that have a specific type and are located inside a user-defined shell ("neighborhood") around a specified QM zone (not including link atoms) as belonging to the selected embedding zone At present, PUPIL allows two concentric embedding zones labeled EMB1 and EMB2. Thus, all the classical particles that

(i) have a specific atom type defined in the "ini unit type" column, *and*

(ii) are separated from any particle belonging to the "QZONE" (quantum zone) by a distance within the limits defined by the "rMin" and "rMax" column,

will be assigned to the requested embedding zone (either EMB1 or EMB2). During the QM calculation, the particles assigned to this zone will be replaced by a quantum atom of type defined by the "End unit type" column, positioned at a distance defined by the "New Distance" column to the closest QM atom. The "New Distance" argument is optional. However, this argument is important when this rule is used to describe *link atoms* (see below). If this argument is not provided, the quantum atom occupies the same position as the classical atom being replaced.

Link atoms. This rule can be used to assign *link atoms* to the quantum zone. In this case, the initial atom type and the initial distance will be set in order to find the correct pair of link atoms. The user has to assign a new bond distance for the link atom; otherwise it will be considered as a normal embedding particle. The way that the GUI distinguishes between usual quantum particles and a link pair description is through the "New Distance" column in the neighboring rules (last column, see Fig. 4). The usual quantum particles have no new distance assigned, while the link pair description will have a new distance between the particles involved in the link pair, introduced by the user.

For example, the quantum zone depicted in Figure 1 (right hand side), representing the active site of the *Tripanosoma cruzi trans*-sialidase enzyme, shows the enzyme's substrate (center) surrounded by three enzyme residues. Each of those residues is connected to two other residues in the enzyme; thus, there are six link atoms in total. The screenshot in Figure 4 shows the assignment of those six link atoms. Since the residues connected to the quantum zone are neither completely quantum nor completely classical, all remaining atoms of those residues

must be specified in the upper left-hand area of the dialogue box (direct atom type assignment) and assigned to the classical zone “EMB1”. This distinguishes them from the remaining classical atoms, which will be assigned later on a per-residue basis to the classical zone “EMB2”.

– *Direct residue type assignment*

This rule is similar to the “direct atom type assignment”, except that the user selects whole residues instead. When the MD CU has an option to describe sets of atoms, (e.g., “molecules” in DL_POLY or “residues” in AMBER), PUPIL groups those sets as residues. A direct relationship is established by default between any classical residue (defined by its residue number) and its associated quantum particles. The mapping rules between the classical kind of atoms belonging to a specific residue and the quantum kind of atoms must be defined by the user (see Section 4.1.3).

In the *trans*-sialidase example shown, the three residues comprising the active site and the two residues comprising the substrate are wholly within the quantum zone, and can easily be assigned in PUPIL using this rule, as shown in Figure 4.

– *Neighboring residue type assignment.*

This is a more general rule to map classical particles belonging to a specific residue to quantum particles based on the distance between the residue center of mass and the nearest, already-assigned) quantum particle. This rule does not allow assignment of link atoms, thus the New Distance option is deactivated. This field can be used to assign any particles that have not yet been defined in the previous boxes. The assignment is done by residue type and distance. Thus, any particle that:

(i) belongs to one of the residue types listed in the “Ini unit type”, *and*

(ii) falls within the distance range defined by “rMin” and “rMax”, *and*

(iii) has not been defined in any of the previous rules,

is assigned to the embedding zone designated in the “zone” column.

In the present example, we defined all possible residues from *trans*-sialidase in that region, and the atoms will be assigned to the “EMB2” zone, to differentiate from the residues directly bonded to the QM residues, which have been assigned to the “EMB1” zone.

4.1.2.3 QM Applications Currently Implemented

The QM force generation applications currently implemented to run with the PUPIL system are:

– *MNDO97*

A computational chemistry package that uses semi-empirical quantum chemistry methodology to obtain quantum forces. The user may choose to use the conventional parameters provided in the package or to use the Transfer Hamiltonian (TH) parameterization via an external file where the TH parameters are stored.

This package works with PUPIL in Start-Stop (SS) mode. At each MD step, the PUPIL Manager writes a modified input file with the corresponding quantum coordinates and executes the MNDO97 binary. The MNDO option file has to be a formatted file following the MNDO97 standards (see the MNDO manual). That file is parsed by PUPIL to obtain needed information. If the user wants to modify the Hamiltonian parameters, the *fort.14* file must be added as an *External Parameters file*.

– *SIESTA*

A computational chemistry and materials package that implements the density functional method to obtain quantum forces. It has a specific interface made to work with the PUPIL system in both Start-Stop and CycleQM modes (see reference **[1]**, sections 2.4.2 and section 2.4.3). The main input file (*.fdf*) has to be specified and the remaining files (*.psf*) must be added in a general list box with the label "Other required files:"(See bottom Figure 3.) After compilation of the SIESTA source code, patched to include the CycleQM capability, the *siesta* binary recognizes a new keyword in the *.fdf* input file that controls the *CycleQM* behavior. Thus, to activate CycleQM the user must include the following line in the *.fdf* file:

```
MultiScale   .true.
```

– *Gaussian 03*

A general and widely used computational chemistry package that has many QM methods for generating quantum forces. This package works with the PUPIL system in Start-Stop (SS) mode (see reference **[3]**).

Version Note*: The force correction method described here is to be superseded in the next minor revision of PUPIL.*

Classical (embedding) particles are represented in Gaussian as immovable point charges through the CHARGE and NOSYMM keywords. PUPIL automatically provides these keywords to Gaussian, along with a list of the classical particles and their charges. Gaussian ignores the electrostatic force exerted on the classical particles by the QM region. Therefore, for a multi-scale simulation, a quantum zone – point charge force correction is necessary (see Section 4.1.2.1).

The current force correction method uses the program *cubegen*, which must be provided in the "Other required files" list (shown in Figure 3 as "Add the input files calculation"). This program calculates a charge grid representing the approximate charge distribution of the QM region as a large number of point charges on a regularly spaced grid. PUPIL calculates the force on the classical particles using these grid points, including only the points whose absolute value is greater than the charge density threshold (see Figure 3).

While this process is computationally intensive and time-consuming, significant speed-up may occur if PUPIL is run as several java threads. We recommend the use of as many Java threads as processors allocated to PUPIL on the computer (or on the head node if using several machines over MPI).

4.1.2.4 Molecular Dynamics (MD).

The CU that controls the time evolution in the simulation is the Molecular Dynamics unit. At each MD step, the MD program asks PUPIL for the forces on the quantum particles, and those forces are evaluated through the PUPIL interface. There is only one common parameter to set for all the MD Calculation Units.

– *MD Steps to extract result.*

  This field determines the frequency (in MD steps) at which the PUPIL Manager will take a snapshot of the system and store all the system coordinates and other internal variables in memory to dump later on into the *output.xml* or other intermediate files.

4.1.2.5 MD Applications Currently Implemented

The MD packages that currently interface with PUPIL are DL_POLY and AMBER 10 (See sections 2.4.2):

- *DL_POLY*

  The interface included in this package connects with the corresponding PUPIL library in serial execution. A simple force difference correction is included (see reference **[1])** where the difference between classical and quantum forces in the quantum zone is considered constant for a predefined number of classical steps. Any improvement and modification of the physics treatment of the forces and energy obtained beyond this approximation from the quantum mechanical computation is the responsibility of the user.

- *AMBER10*

  The Amber10 source code has the modifications needed to interface properly with PUPIL in a standard QM/MM scheme (see reference **[3])**. The user must build (or have available) the corresponding *sander.PUPIL* binary; for instructions, see Section 2.4.2.3.

  The Amber input file does not know about the QM/MM manager. The QM/MM controls in Amber should *not* be invoked because all the QM/MM directives will be introduced through PUPIL packages and QM worker input files:

  - A correction to the forces on point charges must be applied (see reference [3]) (Section 4.1.2 *QZ-PC correction for PC*)
  - The quantum and embedding zones are defined through the PUPIL Graphical User Interface, as well as the link atoms. (Section 4.1.2)
  - The method and level of approximation for the QM calculation should be specified in the QM package input file (*e.g.*, a Gaussian input file). This file is loaded into the GUI and incorporated into the PUPIL XML input.

4.1.2.6 Domain Identification (DI)

Partitioning of the system into regions of quantum and classically generated forces can be done using a set of simple rules (discussed above) or through an external program that

analyzes the system variables to determine where the quantum zone is located. In the PUPIL architecture, external programs that perform this function are called Domain Identification (DI) CUs. The user has the option to call a DI CU several times during the simulation. This opens the possibility of a dynamic treatment for the quantum zone.

- *Steps to Dom. Ident.*

  This parameter tells the Manager how many MD steps are to be taken between two DI calls.

Version Note*: At present no DI interfaces are provided.*

4.1.3 KeyMM/KeyQM Mapping

Note: This mapping is applied <u>only</u> to the atoms not defined in the embedding rules (4.1.2.2) described above.

As discussed in the previous sections, the parsers in the PUPIL GUI extract the kinds of atoms associated with the classical (*KeyMM*) and the quantum (*KeyQM*) systems. A simple mapping between the two kinds of particles is done by the PUPIL system using the atomic number, as shown in **¡Error! No se encuentra el origen de la referencia.**. This default solution may not be sufficiently general for all user needs. For this reason the KeyMM/KeyQM mapping panel allows the user to change the default mapping between classical and quantum particle identifiers.

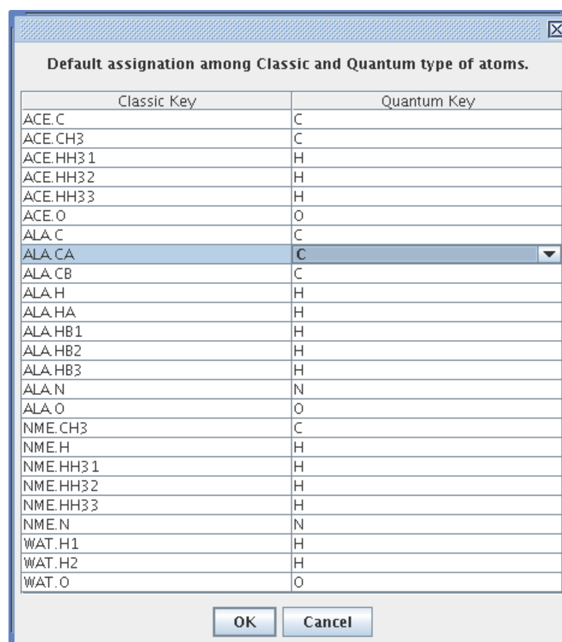
The key used in the mapping may be different depending on internal details of the CU in which the key will be used. The basic convention is as follows:

– *Classical particles: {residue}.{atom}*

  The partition between residue and classical particle (e.g. Amber uses WAT.H1, ALA.CA …) or molecule and atom (e.g. DL_POLY uses SILICON.SI) will be kept in the keyMM and extracted from the input files. To define a custom partitioning of the classical system, a more specific mapping between classical and quantum kind of atoms is allowed.

– *Quantum particles: [PC.][{residue}.]{atom}*

  The quantum keys come from two sources: The first source is the QM CU input files. They yield the user-defined quantum atoms and the point charges (PC) associated with any classical particle from the MD input files. All point charges



Classic Key	Quantum Key
ACE.C	C
ACE.CH3	C
ACE.HH31	H
ACE.HH32	H
ACE.HH33	H
ACE.O	O
ALA.C	C
ALA.CA	C
ALA.CB	C
ALA.H	H
ALA.HA	H
ALA.HB1	H
ALA.HB2	H
ALA.HB3	H
ALA.N	N
ALA.O	O
NME.CH3	C
NME.H	H
NME.HH31	H
NME.HH32	H
NME.HH33	H
NME.N	N
WAT.H1	H
WAT.H2	H
WAT.O	O

Figure 5. Default KeyMM/KeyQM Mapping

have by default a "PC." prefix before the normal key (i.e. PC.0.36). They will also have a residue field if it is supplied by the MD input file (i.e. PC.ALA.CA or PC.SILICON.SI).

Warning! *Every time that the user parses the QM input files, a new mapping between keyMM and keyQM is created automatically from scratch and all the mapping data stored previously is discarded. After each and every parsing, the user must remap all the key associations not automatically generated by default or by the embedding rules. A sanity check always should be done to the KeyMM/KeyQM default mapping.*

The default PUPIL mapping works in the following way: Every pair KeyMM/KeyQM mapping is assigned by atomic number first. *If several KeyQM are defined for the same atomic number, the first KeyQM in alphabetical order is assigned.* For example, from *QMAtomLabel.BasisSet* with Gaussian03 inputs, one could have hydrogens with two basis sets: H.BS1 and H.BS2. Or there could be several MM labels for a single atomic weight. The result of this assignment rule (atomic number, then alphabetic) can be incorrect, depending on the particle labeling and the user's intentions. The user must check this mapping and manually edit it if needed. Use the KeyMM/KeyQM PUPIL window interface (see **¡Error! No se encuentra el origen de la referencia.**).

4.1.4 Show Simulation Tree

This panel has the purpose of showing the internal memory structure of the PUPIL Domain and all its values. It should be the same structure as the XML file produced by the GUI with only minor naming differences. The panel allows the user to see which values are stored in memory and analyze the parsed commands. Another way to obtain this same information is to view the XML file using a general browser.

4.2 Results

There is an option to tell the PUPIL system to store intermediate results from the simulation. This capability is a little primitive in this release, but will be developed in future releases. The basic idea is to monitor the results of the simulation. That could be done through the output files of each CU or through a log file from data being transmitted between CU workers. The option that is implemented in this release allows extraction of some data from the QM CU when it works in Start-Stop mode. Data extraction in *CycleQM* mode is not implemented yet because of its significant performance impact.

4.2.1 QM Simulation Summary

This GUI panel (Figure 6) allows the user to obtain a rapid view of the quantum energy obtained at each step. The data can be exported to a *.csv (Comma Separated Value) file format easily with the “Write file” button option situated at the bottom of the panel.

4.2.2 Extract xmol file

The coordinates of the quantum zone are stored at intervals depending on the value of *MD Steps to extract result* (See Molecular Dynamics in section 4.1.2). With this option, the coordinates can be stored in XML format.

SIMULATION RESUME

Iteration	Total Energy
0	-971.459767
1	-971.45522
2	-971.450328
3	-971.445238
4	-971.439887
5	-971.434288
6	-971.428221
7	-971.421817
8	-971.414995
9	-971.40775
10	-971.400053
11	-971.391961
12	-971.383388
13	-971.374398
14	-971.364983
15	-971.355027
16	-971.344715
17	-971.333847
18	-971.322564
19	-971.310752
20	-971.29893
21	-971.286953
22	-971.274838
23	-971.262514
24	-971.249684

Write File Cancel

Figure 6. Results

5. *XML SIMULATION FILE*

The XML (E**X**tensible **M**arkup **L**anguage) format was designed to describe and store structured data. Simulation data typically is very structured, with complex relationships among input files of CUs, simulation parameters, intermediate and final results and data parsed and/or extracted from CU input files. Therefore, a consistent and comprehensive way to store all that information is to use an XML file. The PUPIL GUI helps the user build this simulation file.

In this section we list the principal XML elements included in the input/output Simulation file.

5.1 The SIMULATIONROOT Element

The main element is the root element (SIMULATIONROOT). It has four sub-elements. Three of them are described within this subsection; a fourth, the SIMULATION element, is described below (section 5.2).

5.1.1 The ATOMDICTIONARY Element

This sub-element stores all the types of quantum atoms that PUPIL holds. The identification for the kind of atom is a serial number <*idUnit*>, but there is a direct relationship with the *keyQM* that is parsed from the QM CU input file and stored as a <*label*> element.

5.1.2 The RESIDUEDICTIONARY Element

This sub-element stores all the types of residues needed in PUPIL for the calculation. The internal identification for the kind of residue is a serial number, marked as <*idUnit*>, but there is a direct relationship with the label, which is parsed from the MD Calculation Unit input file and stored as a <*label*> element. Each residue stores also all the *keyMM* particles belonging to it. Every atom particle belonging to a given residue must have a different *keyMM*.

5.1.3 The KEYMM Element

This sub-element stores the default mapping between the *keyMM* and the *keyQM*. The *keyMM* values are obtained from the MD CU input files, and the keyQM values from the QM CU input files.

5.2 The SIMULATION Element

The simulation element is organized as a number of jobs. Each job belongs to a specific CU and stores all the variables necessary to run it. The job is stored as a nested sequence of records from bottom upward, in which the most specialized records contain the more general records. For instance, an AMBERMDJOB element contains an MDJOB element, which in turn contains a JOB element. First we find the more specialized values and, as we go down the tree branches, we find the more general stored values.

For example, an AMBERMDJOB element is one of the most specialized elements from the SIMULATION element, containing an MDJOB sub-element and a number of other sub-elements specific to AMBER calculations. An MDJOB sub-element contains a JOB sub-element and a number of other sub-elements common to MD jobs in general. Finally the JOB sub-element of MDJOB contains elements for storing the most general simulation variables. An example of a simulation XML file is at the end of this document, in Section 5.3.

Within a SIMULATION, each job corresponds to a User Package that will be part of the multi-scale simulation. There is a general rule for job names, which must be respected in order to avoid internal problems in the execution of the Manager and the PUPIL workers. All job names have a common root *JOB* and a pair of prefixes defined by protocol. First, a generic prefix must be added; this is one of *MD*, *QM* or *DI*, depending on whether the package acts as a Molecular Dynamics, Quantum Mechanics or Domain Identification CU, respectively. This is prefixed in turn by the name of the specific User Package (such as AMBER) that will be used in the simulation with its PUPIL worker. Thus, the User Package name may be found at the beginning of the element name. An example of such an element name is AMBERMDJOB. This prefixing protocol is independent of the way that the CU will interact with PUPIL (Start-Stop or CycleQM mode).

5.3 Example data.xml file

This section presents an incomplete data.xml file, containing entries suitable for an MD simulation using AMBER.

```
<?xml version="1.0" encoding="utf-8" ?>
<SIMULATIONROOT>
    <ATOMDICTIONARY>
        ...
        <ATOM>
            <UNIT>
                <idUnit>4</idUnit>
                <label>O</label>
                <mass>15.9994</mass>
                <charge>0.0</charge>
            </UNIT>
            <atomicNum>8</atomicNum>
        </ATOM>
        ...
    </ATOMDICTIONARY>
    <RESIDUEDICTIONARY>
        ...
        <RESIDUE>
            <UNIT>
                <idUnit>1</idUnit>
                <label>ACE</label>
                <mass>56.046</mass>
                <charge>0.0</charge>
            </UNIT>
            <keyMMAtoms>
                <atomUnit>ACE.HH31</atomUnit>
                <atomUnit>ACE.CH3</atomUnit>
                <atomUnit>ACE.HH32</atomUnit>
                <atomUnit>ACE.HH33</atomUnit>
                <atomUnit>ACE.C</atomUnit>
                <atomUnit>ACE.O</atomUnit>
            </keyMMAtoms>
```

```
    </RESIDUE>
    ...
</RESIDUEDICTIONARY>
<KEYMM>
    ...
    <keyMM>
        <key>ACE.O</key>
        <QMkey>4</QMkey>
    </keyMM>
    ...
</KEYMM>
<SIMULATION>
    ...
    <jobs>
        <AMBERMDJOB>
            <MDJOB>
                <JOB>
                    <idJob>AmberMDJob3</idJob>
                    <exe>
                        <path>../../../bin/sanderLinux</path>
                    </exe>
                    <nResources>0</nResources>
                    <files>
                        <ITRFILE>
                            <idFile>mdin</idFile>
                            <path>../data/mdin</path>
                            <sections></sections>
                        </ITRFILE>
                        <ITRFILE>
                            <idFile>ala3.parm7</idFile>
                            <path>../data/ala3.parm7</path>
                            <sections></sections>
                        </ITRFILE>
                        <ITRFILE>
                            <idFile>ala3.inpcrd</idFile>
                            <path>../data/ala3.inpcrd</path>
                            <sections></sections>
                        </ITRFILE>
                    </files>
                    <coordinates>
                        ...
                    </coordinates>
                    <residues>
                        ...
                    </residues>
                </JOB>
            <stepsToShow>2</stepsToShow>
            </MDJOB>
        <amberfiles>
        <ITRFILE>
        <idFile>mdin</idFile>
        <path>../data/mdin</path>
        <sections></sections>
        </ITRFILE>
        <ITRFILE>
        <idFile>prmtop</idFile>
        <path>../data/ala3.parm7</path>
        <sections></sections>
        </ITRFILE>
        <ITRFILE>
        <idFile>inpcrd</idFile>
        <path>../data/ala3.inpcrd</path>
        <sections></sections>
        </ITRFILE>
        </amberfiles>
        </AMBERMDJOB>
```

```
        ...
        </jobs>
    </SIMULATION>
</SIMULATIONROOT>
```