APBS 0.5.1 User Guide

Adaptive Poisson-Boltzmann Solver

Nathan A Baker

Washington University in St. Louis
Department of Biochemistry and Molecular Biophysics
Center for Computational Biology

Table of Contents

Chapter 1. Introduction

APBS is a software package for the numerical solution of the Poisson-Boltzmann equation (PBE), one of the most popular continuum models for describing electrostatic interactions between molecular solutes in salty, aqueous media. Continuum electrostatics plays an important role in several areas of biomolecular simulation, including:

- simulation of diffusional processes to determine ligand-protein and protein-protein binding kinetics,
- implicit solvent molecular dynamics of biomolecules,
- solvation and binding energy calculations to determine ligand-protein and protein-protein equilibrium binding constants and aid in rational drug design,
- and biomolecular titration studies.

APBS was designed to efficiently evaluate electrostatic properties for such simulations for a wide range of length scales to enable the investigation of molecules with tens to millions of atoms.

APBS uses PMG to solve the Poisson-Boltzmann equation numerically. PMG is developed and maintained by the Holst Research Group at UC San Diego, and is designed to solve the nonlinear Poisson-Boltzmann equation and similar problems with linear space and time complexity through the use of box methods, inexact Newton methods, and algebraic multilevel methods. More information about PMG may be found at http://www.fetk.org.

APBS also uses FEtk to solve the Poisson-Boltzmann equation numerically. FEtk is developed and maintained by the Holst Research Group at UC San Diego, and is designed to solve general coupled systems of nonlinear partial differential equations accurately and efficiently using adaptive multilevel finite element methods, inexact Newton methods, algebraic multilevel methods. More information about FEtk may be found at http://www.fetk.org.

Financial support. The development of APBS has been supported financially by:

- National Institutes of Health (grant GM069702-01)
- National Partnership for Advanced Computational Infrastructure
- National Biomedical Computation Resource

Citing APBS. Please acknowledge your use of APBS by citing:

> Baker NA, Sept D, Joseph S, Holst MJ, McCammon JA. Electrostatics of nanosystems: application to microtubules and the ribosome. *Proc Natl Acad Sci USA* 98, 10037-10041, 2001.

Citing FEtk and PMG. Please acknowledge your use of PMG and FEtk by citing:

> M. Holst and F. Saied, Multigrid solution of the Poisson-Boltzmann equation. *J. Comput. Chem.*, 14 (1993), pp. 105-113.
>
> M. Holst and F. Saied, Numerical solution of the nonlinear Poisson-Boltzmann equation: Developing more robust and efficient methods. *J. Comput. Chem.*, 16 (1995), pp. 337-364.
>
> M. Holst, Adaptive numerical treatment of elliptic systems on manifolds. *Advances in Computational Mathematics*, 15 (2001), pp. 139-191.
>
> R. Bank and M. Holst, A New Paradigm for Parallel Adaptive Meshing Algorithms. *SIAM Review*, 45 (2003), pp. 291-323.

Contributing authors. APBS was primarily written by Nathan Baker during his graduate work with J. Andrew McCammon and Michael Holst and extensively developed over the subsequent years. APBS uses several libraries written by Mike Holst and members of the Holst group, including: PMG (multigrid solver for Cartesian mesh discretization), FEtk (provides finite element framework, error estimators, and solvers), and MALOC (hardware abstraction library for code portability). Additionally, a number of people have made important contributions to enhance APBS functionality and usability. The full author list (in alphabetical order) is:

- Nathan A. Baker

  Primary author
- Steve Bond

  Contributor: FEtk library compatibility and Vopot functions
- Larry Canino

  Contributor: solvent accessibility function modification
- Todd Dolinsky

  Developer and contributor: Python wrappers, PDB2PQR, output logging, examples, and other tools
- Adrian Elcock

  Contributor: parameter files
- David Gohara

  Developer
- Michael J. Holst

  Advisor and author of several routines scattered about APBS, and author of PMG (the multigrid library used by this code), author of MALOC (hardware abstraction library used by this code), author of FEtk (finite element library incorporated in developmental versions of this

code)

- Adrian Kaats

  Contributor: multivalue.c

- Robert Konecny

  Contributor: parameters, CHARMM FORTRAN interface

- Jung-Hsin Lin

  Contributor: OpenDX to MOLMOL conversion

- Chiansan Ma

  Contributor: format conversion scripts

- J. Andrew McCammon

  Advisor, financial and equipment support, functionality suggestions

- Jens Nielsen

  Contributor: PDB2PQR

- Jay Ponder

  Contributor: TINKER Interface

- Michael Schnieders

  Contributor: Polarizable atomic multipole routines, TINKER interface

- David Sept

  Contributor: scripts and tools, VMD compatibility, general suggestions about functionality

- Tongye Shen

  Contributor: finite element mesh generation

- Justin Xiang

  Contributor: generalized Born Python utilities

Chapter 2. Installation

Table of Contents

The following sections outline the installation of APBS on a generic UNIX platform as well as installation instructions for various specific machines.

2.1. Availability

The latest version of APBS can always be found at http://apbs.sourceforge.net. APBS is available in both source code form and binaries for a variety of architectures.

2.2. Binary installation

We currently offer binaries for the RedHat Linux platform on a variety of architectures as well as command line binaries for WinXP and Mac OS X. Binaries can be downloaded from the APBS download page. For all other systems, please install from source on your particular platform and feel free to contact the APBS users mailing list for more help and/or to request a binary for that system.

2.3. Source installation

If you were unable to find the binary pacakge for your system, or would like to compile APBS yourself, you'll need to read the instructions in this section.

2.3.1. Prerequisites

In order to install APBS from the source code, you will need:

- C and Fortran compilers
- The APBS source code (see above)

> **Note**
> Note that the Holst group MALOC source code is now provided with APBS to facilitate installation.

It may also be useful to have:

- A version of MPI (try MPICH) for parallel jobs.

  > **Note**
  > MPI isn't strictly necessary if the `async` option is used.

- A vendor-supplied version of BLAS for optimal performance. APBS will attempt to locate common BLAS libraries during the configuration process.

2.3.2. Tested systems

Source-code-based installation has been tested on the following systems

- AMD 64 (Opteron) running Fedora Core 6 Linux
- IBM Power4 running RedHat Enterprise Linux 3
- SGI Itanium2 (Altix) running IRIX/Linux
- Apple PowerPC running Mac OS X
- Apple x86_64 running Mac OS X

However, the installation procedure is rather generic and generally works on most UNIX-based systems. System-specific installation notes and caveats are provided in Section 2.3.5, “System-specific notes”.

2.3.3. Preparation for installation

In what follows, I'll be assuming you're using bash, a shell available on most platforms (UNIX and non-UNIX).

2.3.3.1. Compiler variables

First, please look at Section 2.3.5, “System-specific notes” section of this document for appropriate compiler flags, etc. to be set via pre-configuration environmental variables. This section outlines generic installation procedures with default compilers and options.

2.3.3.2. Installation directories

There are a few directories you'll need to identify prior to installation. The first, which we'll call *`APBS_SRC`*, will contain the APBS and MALOC source code. This directory can be deleted after installation, if you wish. The second directory will be the permanent location for APBS and MALOC; we'll call this *`APBS_PREFIX`*. If you have root permission, you could pick a global directory such as `/usr/local` for this; otherwise, pick a directory for which you have write permission. The following commands set up the directories and environmental which point to them:

```
$ export APBS_SRC=/home/soft/src
$ export APBS_PREFIX=/home/soft
$ mkdir -p ${APBS_SRC} ${APBS_PREFIX}
```

2.3.3.3. Unpacking the source code

You're now ready to unpack the source code:

```
$ cd ${APBS_SRC}
$ gzip -dc apbs-0.5.1.tar.gz | tar xvf -
```

2.3.4. Configuring, compiling, and installing

Before compiling/installing APBS, you need to configure with the autoconf `configure` script. You can examine the various configure options with the `--help` option. For many platforms, no options need to be specified. Therefore, most users *who want single-CPU (not parallel) binaries* can configure as follows:

```
$ cd ${APBS_SRC}/apbs
$ ./configure --prefix=${APBS_PREFIX}
```

Other configuration options, including the compilation of parallel binaries and the use of machine-specific compilers and Python usage, are discussed in Section 2.3.5, “System-specific notes”.

Assuming all has gone well with the configuration (you'll generally get an error message if configuration fails), you're ready to compile

```
$ make all
```

and install APBS:

```
$ make install
```

The installation will create several directories under *`${APBS_PREFIX}`*:

- `bin`, where the main `apbs` binary resides
- `examples`, which contains a number of examples and test cases for APBS
- `include`, which contains header files for using APBS libraries with other applications
- `lib`, which contains library files for using APBS libraries with other applications
- `tools`, which contains a number of "helper" applications for use with APBS.

At this point you are ready to use APBS; either by calling the binary directly or adding the above directory to your path. As mentioned above, there are also several tools provided with APBS that remain in the APBS directory; these are described in later portions of this manual. You may wish to copy these to a global location (or the same place as your APBS binary) at this time.

2.3.5. System-specific notes

> **Important**
> If you have tips or tricks on improving APBS performance and/or installation on your machine, please let us know!

This section provides tips on compiling APBS on specific platforms and outlines some of the basic options available for parallel execution, Python linkage, etc. Note that many aspects of this section have changed from previous releases. In particular, APBS now tries to detect the optimal compilers and BLAS libraries on most systems without user intervention.

As described above, the default configure-make-install procedure is

```
$ ./configure --prefix=${APBS_PREFIX}
$ make
$ make install
```

The configure script includes a number of generic options to manually set default compilers, link behaviors, preprocessors, etc. These can be reviewed by running

```
$ ./configure --help
```

2.3.5.1. Python support

Python libraries and related tools can be enabled at configure-time. Currently, Python libraries compile on most Linux and Mac systems. Other systems are untested. The configure-make-install procedure is:

```
$ ./configure --enable-python --prefix=${APBS_PREFIX}
$ make
$ make install
```

2.3.5.2. Parallel (MPI) support

APBS uses MPI for parallel execution. In general, MPI support requires informing the APBS configure script about:

- MPI compiler options. For most MPI implementations, you simply need to set the *CC* and *F77* variables to point to the MPI-savvy C and FORTRAN compilers. However, it is occasionally necessary to manually specify compiler options by setting *CFLAGS*, *CPPFLAGS*, *FFLAGS*, and *LDFLAGS* before configuration.
- MPI library/header file locations. As outlined below, paths to the library and header files for LAM and MPICH implementations of MPI can be specified with the `--with-lam`, `--with-mpich`, or `--with-mpich2` configure options. Other MPI implementations will require the *CFLAGS*, *CPPFLAGS*, *FFLAGS*, and *LDFLAGS* variables to be set correctly before configuration to locate the required headers and libraries.

2.3.5.2.1. LAM MPI

It is important that you enable FORTRAN support and use the same compilers you will use to compile APBS when installing/compiling LAM MPI. For example, if your C compiler is set in the environmental variable *${CC}* and your FORTRAN compiler is set in the environmental variable *${F77}*, then you should configure *LAM* with the command

```
$ ./configure --prefix=${MPI_PREFIX} --with-fc=${F77}
```

Let *${MPI_PREFIX}* be an environmental variable pointing to the directory where LAM MPI is installed. In other words, `${MPI_PREFIX}/lib` should contain the LAM MPI libraries and `${MPI_PREFIX}/include` should contain the LAM MPI header files. The configure-make-install procedure is then

```
$ export CC=${MPI_PREFIX}/bin/mpicc
$ export F77=${MPI_PREFIX}/bin/mpif77
$ ./configure --with-lam=${MPI_PREFIX} --prefix=${APBS_PREFIX}
$ make
$ make install
```

This procedure was tested with LAM MPI 7.2.1.

2.3.5.2.2. MPICH1

It is important that you enable FORTRAN support and use the same compilers you will use to compile APBS when installing/compiling MPICH1. For example, if your C compiler is set in the environmental variable *${CC}* and your FORTRAN compiler is set in the environmental variable *${F77}*, then you should configure *MPICH1* with the command

```
$ ./configure --prefix=${MPI_PREFIX} ... --enable-f77
```

where `...` denotes other machine-specific options such as `--with-device=ch_p4` or `--with-arch=LINUX`.

Let *${MPI_PREFIX}* be an environmental variable pointing to the directory where MPICH1 is installed. In other words, `${MPI_PREFIX}/lib` should contain the MPICH1 libraries and `${MPI_PREFIX}/include` should contain the MPICH1 header files. The configure-make-install procedure is then

```
$ export CC=${MPI_PREFIX}/bin/mpicc
$ export F77=${MPI_PREFIX}/bin/mpif77
$ ./configure --with-mpich=${MPI_PREFIX} --prefix=${APBS_PREFIX}
$ make
$ make install
```

This procedure was tested with MPICH1 1.2.7p1.

2.3.5.2.3. MPICH2

It is important that you enable FORTRAN support and use the same compilers you will use to compile APBS when installing/compiling MPICH2.

Let *${MPI_PREFIX}* be an environmental variable pointing to the directory where MPICH2 is installed. In other words, `${MPI_PREFIX}/lib` should contain the MPICH2 libraries and `${MPI_PREFIX}/include` should contain the MPICH2 header files. The configure-make-install procedure is then

```
$ export CC=${MPI_PREFIX}/bin/mpicc
$ export F77=${MPI_PREFIX}/bin/mpif77
$ ./configure --with-mpich2=${MPI_PREFIX} --prefix=${APBS_PREFIX}
```

```
$ make
$ make install
```

This procedure was tested with MPICH1 1.2.7p1.

2.3.5.3. FEtk support

In order to enable support for the finite element features in APBS, you must compile it against the FEtk libraries. *Please use the same compilers for both APBS and FEtk.* Let *${FETK_PREFIX}* be an environmental variable pointing to the directory where FEtk was installed. In other words, `${FETK_PREFIX}/lib` should contain the FEtk machine-specific library directories and `${FETK_PREFIX}/include` should contain the FEtk header files. You'll first need to identify the appropriate library directory for your system in `${FETK_PREFIX}/lib`. For the purposes of this example, let's suppose this directory is `${FETK_PREFIX}/lib/x86_64-unknown-linux-gnu`. The configure-make-install procedure for APBS is then

```
$ ./configure --with-fetk-include=${FETK_PREFIX}/include --with-fetk-library=${FETK_PREFIX}/lib/x86_64-unknown-linux-gnu --prefix=$
$ make
$ make install
```

2.3.5.4. Windows

We are happy to now provide native APBS command line binaries for Windows. The binary is probably the best option available, but if you would still like to compile your own binaries you will need to use either the Cygwin or MinGW environments. Binaries compiled under Cygwin tend to require Cygwin DLLs and thus can only be run on systems with Cygwin. Performance for the Windows binaries and all compiled systems will be fairly mediocre as they depend on the GNU compilers.

If you do choose to use Cygwin and compile your own code, compilation should be rather straightforward.

2.3.5.5. Macintosh

We are happy to now provide a Mac install package for Mac OS 10.4 (Tiger). Unfortunately this is the only binary for Mac that we have available, so users on OS 10.3 may have to compile binaries for themselves - you may want to examine the apbs-users mailing list which has a number of threads which discuss installation on Mac OS platforms. Alternatively you can try using Fink for the installation - please see Bill Scott's excellent guidelines at http://chemistry.ucsc.edu/~wgscott/xtal.

A few notes about compiling on Macintosh:

1. It has become apparent from the mailing lists that some "packages" of the GNU development software available for MacOS contain different major versions of the C and FORTRAN compilers. This is very bad; APBS will not compile with different versions of the C and FORTRAN compilers. If you use GCC 4.0, for instance, gfortran 4.0 will work while g77 3.3 will not. If you see link errors involving "restFP" or "saveFP" this is most likely the cause.
2. In gcc 4.0 (included in Xcode 2.0 and higher) the -fast option turns on the -fast-math flag. This flag optimizes by using rounding, and thus can lead to inaccuate results and should be avoided.
3. As it stands now the autoconf script does not support using the native vecLib framework as an architecture-tuned BLAS replacement. In testing there were only slight timing improvements over using the MALOC-supplied BLAS as it is.
4. We have had success using IBM's XLF for Mac in conjunction with GCC 4.0, although the corresponding XLC compilers do not seem to work under Tiger.

Finally, Dave Gohara has prepared a nice tutorial on building APBS from within Xcode to take advantage of its development and debugging features.

Chapter 3. Overview

Table of Contents

This chapter gives an overview of the binaries, tools, etc. distributed as part of the APBS software package. It is organized by directory; later chapters provide a more in-depth description on tools specific to particular applications.

3.1. Invocation

As mentioned in the Installation intructions, the main APBS binary is installed in `${FETK_PREFIX}/bin/${prefix}/` where `${prefix}` is a machine-specific directory. Of course, you can move the binary to any directory you choose. APBS is invoked with a very simple syntax:

`apbs` [options] input-file

Command line options include:

`--outputfile=`*`name`*

Sets the output logging path (as described in the output logging section of the manual) to *`name`*, or *`name_N`* for parallel runs, where `N` is the processor ID. If `--outputformat` is not specified, flat-file format will be used as the default.

`--outputformat=`*`type`*

Sets the output logging format. Accepted values are:

`flat`

Flat-file format (default).

`xml`

XML format

`--help`

Displays command line usage

`--version`

Displays the current APBS version

`input-file` is an input file with a specific syntax described in Section 4.2, “Input files”. Besides the output files specified in `input-file` and optional logs as specified by use of the `--output-file` command line option, APBS writes data to three additional places:

- Standard output. This will appear on your screen (if you don't redirect it somewhere) and will contain all the basic information about the electrostatics calculation.
- Standard error. This will also appear on your screen (if you don't redirect it somewher) and will contain warnings and error messages.
- The file `io.mc` (or `io.mc_N` for parallel runs, where `N` is the processor ID. This gives you detailed information about the progress of the run with a particular focus on the numerical solver.

3.2. Other tools

APBS contains a number of tools to facilitate the preparation of APBS runs and analysis of the results.

Note
NOTE: In addition to the tools provided with APBS, there are a number of other programs which interoperate with our code. Please see the Other Programs section of this manual for more information.

3.2.1. Parameterization

Unfortunately, the majority of problems encountered during electrostatics calculations arise in process of taking a structure from the Protein Data Bank and transforming into a file that can be used by the APBS software. The PDB2PQR service was orginally developed in conjunction with

Jens Nielsen and Andy McCammon to address these issues. The service has since evolved and has been completely rewritten by Todd Dolinsky and Nathan Baker. PDB2PQR is able to:

- Fill in missing atoms in the PDB (within reason)
- Add hydrogens to the structure to optimize the hydrogen-bonding network.
- Calculate side-chain pKas
- Assign charges and radii according to one of the following force fields: CHARMM23, AMBER02, or PARSE
- Return the results in PQR format
- Generate APBS input files.

Please visit the PDB2PQR home page (http://pdb2pqr.sourceforge.net) for more information, links to available servers, and download options.

Additionally, APBS provides the ability to read plain PDB-format files and assign charges and radii from user-supplied parameter files. These features are described in the READ PARAM command description.

Finally, APBS provides a few other miscellaneous tools for converting and parameterizing structures:

- `tools/conversion/qcd2pqr.awk`

  Convert a QCD file (UHBD format for a molecule) to PQR format.

- `tools/conversion/amber2charmm.sh`

  A script which converts a PDB file with AMBER atom names to a PDB file with CHARMM atom names. Useful for preprocessing files before converting with pdb2pqr.

- `tools/conversion/WHATIF2AMBER.sed`

  A sed script for converting a PDB file with WHATIF atom names to a PDB file with AMBER atom names. Useful for preprocessing files before converting with pdb2pqr. Contributed by Chiansan Ma.

3.2.2. Problem setup

In addition to parameterization of the molecule, there are several common operations which are performed to setup the calculation. This section reviews some of the tools available for these operations. Please note that PDB2PQR (see Section 3.2.1. “Parameterization” above) also prepares APBS input files.

The following scripts help generate or transform APBS input files:

- `tools/manip/psize.py`

  Get the dimensions and center of a molecule in PQR format. Very useful for setting up input files (i.e., grid dimensions, lengths, spacings, etc.) for APBS calculations. Written by Todd Dolinsky and Nathan Baker.

- `apbs/tools/manip/inputgen.py`

  Generate an APBS input file from PQR format data using "suggested" parameters. Also can decouple a parallel calculation into a series of sequential (asynchronous) calculations to be performed on a single processor. Written by Todd Dolinsky and Nathan Baker.

- `tools/mesh/mgmesh`

  List acceptable grid dimensions/multigrid levels combinations for mg-manual calculations. Written by Nathan Baker

3.2.3. Output data processing

The following tools perform typical analyses of the output data, usually in OpenDX format. These scripts are not meant to be comprehensive; instead, they provide templates for users to generate their own tools.

3.2.3.1. Conversion

- `tools/mesh/uhbd_asc2bin`

  Converts UHBD-format grid files from ASCII to binary. Contributed by Dave Sept.

- `tools/mesh/dx2mol`

  Converts OpenDX format data to MOLMOL format. Contributed by Jung-Hsin Lin with bug fixes by Fred Damberger.

- `tools/mesh/dx2uhbd`

  Converts OpenDX format data to UHBD format. Contributed by Robert Konecny.

3.2.3.2. Manipulation

- `tools/mesh/mergedx`

  Merge OpenDX format data from several domains (e.g., from a mg-para calculation into a single file. This program is deprecated (replaced by `mergedx2` and will be removed in an upcoming release. Contributed by Steve Bond.

- `tools/mesh/mergedx2`

  Merge OpenDX format data from several domains (e.g., from a mg-para calculation into a single file while allowing resampling of the data to increase/decrease resolution. This function will eventually replace `mergedx`. Contributed by Dave Gohara.

- `tools/mesh/smooth`

  Apply a very inefficient Gaussian filter to OpenDX format data from APBS. Written by Nathan Baker.

3.2.4. Data visualization

This section describes the data visualization tools provided with APBS. A more complete discussion of the various ways to visualize APBS output is presented in the Visualization section of this manual.

- `tools/visualization/vmd`

  This directory contains scripts which facilitate the visualization of APBS data with VMD.

  > **Note**
  > NOTE: As described in the Visualization section, a much more elegant interface has been developed for APBS and is available from the VMD plugins page.

  The version distributed with APBS was written by Nathan Baker and Dave Sept based on example Tcl scripts by John Stone. The file `loadstuff.vmd` is the command file to be modified to the users' tastes and loaded into VMD. The file `read_dx` contains the Tcl functions needed to read the APBS output.

- `tools/visualization/opendx`

  This directory contains the OpenDX program files (`*.net`) required to visualize APBS data with OpenDX. In particular, one can visualize single-file potential isocontours (`pot.*`), single-file potential data mapped onto molecular surfaces (`potacc.*`), or multiple-file potential data (`multipot.*`).

3.2.5. Solvent accessibility

The main APBS executable calculates molecular volumes, surface areas, and other surface-based properties from PQR-format structural data. Such calculations are often used to determine apolar solvation contributions to binding events, etc. See the new APOLAR keyword for more documentation on this APBS feature.

3.2.6. Coulomb's Law and Generalized Born calculations

These utilities are provided for occasional use and are *definitely not* optimized for speed.

> **Note**
> NOTE: Many of these tools will be incorporated into the main APBS executable during upcoming releases!

3.2.6.1. Coulomb's Law calculations

The program `tools/manip/coulomb` calculates *vacuum* Coulomb law energies from a PQR file. It has a number of options which can be viewed by running the `coulomb` program with no arguments.

3.2.6.2. Generalized Born calculations

The program `tools/manip/born` is a crude, non-optimal, buggy program (are you still reading?!?) for calculating Generalized Born electrostatic energies. This is only intended for hacking and general comparison with Poisson-Boltzmann results.

The Python-based program `tools/python/runGB.py` is a test program designed to calculate generalized Born radii from APBS Poisson-Boltzmann calculations following the general methods of Onufriev A, Case DA, Bashford D. Effective Born radii in the generalized Born approximation: The importance of being perfect. J Comput Chem. 23 (14), 1297-304, 2002. http://dx.doi.org/10.1002/jcc.10126. More information on this program can be obtained by running it from the command line with the `--help` option.

The Python-based program `tools/python/readGB.py` is a test program designed to use radii calculated from `runGB.py` (see above) and print out solvation energies. More information on this program can be obtained by running it from the command line with the `--help` option.

Both of these Python-based programs were written by Justin Xiang.

3.2.7. Eigenvalue analysis

`tools/arpack/driver`

If APBS is linked with ARPACK (see `configure --help`), this routine will perform eigenvalue analyses of matrices produced by APBS.

3.2.8. Python development tools

There are a number of example Python tools and wrappers provided in the tools/python directory. These tools all make use of the APBS SWIG wrappers developed by Todd Dolinsky, Nathan Baker, Alex Gillet, and Michel Sanner. The SWIG wrappers are compiled by default during normal installation. The Python scripts which link to the wrappers (and thereby illustrate their use) include:

- `tools/python/main.py`

  Drop-in replacement for main APBS executable. Only permits sequential runs.

- `tools/python/noinput.py`

  Similar to main.py, but adds the ability to read input files and PQR files as Python strings and return energies and forces as Python lists. This makes it a very useful tool for working with APBS via Python without dealing with a great deal of file I/O.

- `tools/python/vgrid/`

  Python wrappers for Vgrid class to allow OpenDX format file I/O in Python scripts

3.3. Examples and tutorial

The APBS sub-directory `examples` contains several test systems which show how to use APBS for binding energy, solvation energy, and force calculations. The file `examples/README.html` contains descriptions of the test cases and links to anticipated results. Examples can be run and compared to expected results by running `make test` in each example directory.

Additional examples are provided as part of the APBS tutorial (`doc/html/tutorial/`), described in more detail in the Documentation section.

3.4. Documentation

The APBS sub-directory `doc` contains guides for using APBS and developing code based on APBS libraries. The subdirectories include:

- `doc/html/user-guide/index.html`

  HTML-format User Guide

- `doc/html/programmer/index.html`

  HTML-format Programmer Guide

- `doc/html/tutorial/index.html`

  HTML-format APBS tutorial

3.5. Source code

The APBS sub-directory `src` contains the source code for the APBS libraries and main executable. These files are described in more detailed in the Programming section.

Chapter 4. Using APBS

Table of Contents

4.1. Invocation

As mentioned in the Installation intructions, the main APBS binary is installed in `${FETK_PREFIX}/bin/${prefix}/` where `${prefix}` is a machine-specific directory. Of course, you can move the binary to any directory you choose. APBS is invoked with a very simple syntax:

`apbs` [options] input-file

Command line options include:

`--outputfile=`*`name`*

Sets the output logging path (as described in the output logging section of the manual) to *`name`*, or *`name_N`* for parallel runs, where `N` is the processor ID. If `--outputformat` is not specified, flat-file format will be used as the default.

`--outputformat=`*`type`*

Sets the output logging format. Accepted values are:

`flat`

Flat-file format (default).

`xml`

XML format

`--help`

Displays command line usage

`--version`

Displays the current APBS version

`input-file` is an input file with a specific syntax described in Section 4.2, “Input files”. Besides the output files specified in `input-file` and optional logs as specified by use of the `--output-file` command line option, APBS writes data to three additional places:

- Standard output. This will appear on your screen (if you don't redirect it somewhere) and will contain all the basic information about the electrostatics calculation.
- Standard error. This will also appear on your screen (if you don't redirect it somewher) and will contain warnings and error messages.
- The file `io.mc` (or `io.mc_N` for parallel runs, where `N` is the processor ID. This gives you detailed information about the progress of the run with a particular focus on the numerical solver.

4.2. Input files

APBS input files are loosely-formatted files which contain information about the input, parameters, and output for each calculation. These files are whitespace- or linefeed-delimited. Comments can

be added to the input files via the # character; all text between the # and the end of the line is not parsed by APBS. Specific examples of APBS input are described in the Examples section.

> **Tip**
> Please note that there are several tools which help prepare APBS input files based on molecular structures, memory constraints, etc. These tools are described in more detail in Section 3.2.2, "Problem setup".

APBS input files contain three basic sections which can be repeated any number of times:

- **READ**: section for specifying input.
- **ELEC**: section for specifying polar solvation (electrostatics) calculation parameters.
- **APOLAR**: section for specifying apolar solvation calculation parameters.
- **PRINT**: section for specifying summary output.

The APBS input file is constructed from these sections in the following format:

Example 4.1. Template for APBS input file

```
READ
...
END
ELEC
...
END
APOLAR
...
END
ELEC
...
END
APOLAR
...
END
PRINT
...
END
QUIT
```

These sections can occur in any order, however, they are clearly interdependent. For example, **PRINT** requires **ELEC** and/or **APOLAR** while **ELEC** requires one or more **READ** sections. Sections can also be repeated; several **READ** statements may be used to load molecules and multiple **ELEC** or **APOLAR** sections would specify various electrostatics calculations on one or more molecules.

4.2.1. READ statements

`READ` [*keywords*...]
`END`

One of these sections must be present for every molecule involved in the APBS calculation. Molecule and "map" IDs are assigned implicitly assigned for each molecule/map read, based on order and starting at 1. This section has the following keywords:

- `mol` {*format*} {*path*}

  This command specifies the molecular data to be read into APBS. The arguments are:

  format

  The format of the input data. Acceptable flags include:

  pqr

  Molecular data is in PQR format

  pdb

  Molecular data is in PDB format.

  > **Warning**
  > We do not completely follow the PDB format, as specified in the above link. Specifically, we allow general whitespace-, tab-, or

newline-delimited format, thereby permitting the manipulation of molecules with coordinates outside the ±999 range.

Note
Beginning with APBS 0.5.0, the optional use of the chain ID field is now supported in both PDB and PQR formats.

path

The location of the molecular data file.

- `parm` {*format*} {*path*}

  This command specifies the charge and radius data to be used with PDB-format molecule files. The arguments are:

  format

  The format of the parameter file. Acceptable flags include:

  flat

  APBS flat-file parameter format

  xml

  APBS XML parameter format

  path

  The location of the parameter data file.

- `diel` {*format*} {*path-x*} {*path-y*} {*path-z*}

  This command allows APBS to read the dielectric function $\varepsilon(x)$ mapped to 3 meshes shifted by one-half grid spacing in the x, y, and z directions. The inputs are maps of dielectric variables between the solvent and biomolecular dieletric constants; these values are unitless. In general, this command will read dielectric maps written by **write** commands in earlier APBS calculations.

  Note
  NOTE: if you choose this option and have a non-zero ionic strength, you must also include a **read kappa** statement

  Arguments for this command are:

  format

  The format of the dielectric map. Acceptable values include:

  dx

  OpenDX format (see Formats section)

  path-x

  The location of the x-shifted dielectric map file.

  path-y

  The location of the y-shifted dielectric map file.

  path-z

  The location of the z-shifted dielectric map file.

- `kappa` {*format*} {*path*}

  This command allows APBS to read the ion-accessibility function $\bar{\kappa}^2(x)$ mapped to a mesh. The inputs are maps of ion accessibility values which range between 0 and the build Debye-Hückel screening parameter; these values have units of $Å^{-2}$. In general, this command will read kappa-maps written by **write** commands in earlier APBS calculations.

  Note
  NOTE: if you choose this option, you must also include a **read diel** statement

Arguments for this command are:

format

The format of the kappa map. Acceptable values include:

dx

OpenDX format (see Formats section)

path

The location of the kappa map file.

- `charge` {*format*} {*path*}

  This command allows APBS to read the fixed (molecular) charge density function mapped to a mesh. The inputs are maps of charge densities; these values have units of e_c (electron charge) per Angstrom^3. In general, this command will read charge-maps written by **write** commands in earlier APBS calculations. Arguments for this command are:

  format

  The format of the charge map. Acceptable values include:

  dx

  OpenDX format (see Formats section)

  path

  The location of the charge map file.

4.2.2. ELEC statements

`ELEC` [name *id*] {*type*} [*keywords*...]
`END`

This section is the main component for polar solvation calculations in APBS runs. There may be several **ELEC** sections, operating on different molecules or using different parameters for multiple runs on the same molecule. The order of the **ELEC** statement matters (see above); the arguments are:

name *id*

This optional command allows users to assign an alphanumeric string to the calculation to facilitate later operations (particularly in the **PRINT** statements).

type

Specify the type of electrostatics calculation to perform (these are described in greater detail below):

- **mg-auto** for automatically-configured sequential focusing multigrid calculations.
- **mg-para** for automatically-configured parallel focusing multigrid calculations.
- **mg-manual** for manually-configured multigrid calculations.
- **fe-manual** for manually-configured adaptive finite element calculations.
- **mg-dummy** for calculations of surface and charge distribution properties which do not require solution of the PBE.

keywords

Keywords describing the parameters of the electrostatic calculation. These are described in the Keywords section below.

4.2.2.1. Automatic sequential focusing multigrid calculation (mg-auto)

This automatically sets up and performs a string of single-point PBE calculations to "focus" on a region of interest (binding site, etc.) in a system. It is basically an automated version of mg-manual designed for easier use. Most users should probably use this version of **ELEC**.

The keywords for this command (described in more detail in the Keywords section) are listed below. All keywords are required (no default values!) unless otherwise noted: **dime** , **cglen** , **fglen** , **cgcent** , **fgcent** , **mol** , **lpbe** or **npbe** or **smpbe** , **bcfl** , **ion** (optional) , **pdie** , **sdie** , **chgm** ,

usemap (optional) , **sdens** , **srfm** , **srad** , **swin** , **temp** , **calcenergy** , **calcforce** , **write** , **writemat**

4.2.2.2. Automatic parallel focusing multigrid calculation (mg-para)

This calculation closely resembles **mg-auto** in syntax. However, it is basically designed to perform single-point calculations on systems in a parallel focusing fashion. While this method does provide support for decreasing the domain size from a coarse (large) global grid to a fine (smaller) global grid, it should not be used to look at subsets of biomolecules such as titration sites, etc. Such subset calculations require more complicated energy evaluation which is not yet supported by **mg-para**. However, since parallel focusing was designed to provide detailed evaluation of the electrostatic potential on a large scale, such subset calculations are better left to traditional focusing via the **mg-auto** keyword.

> **Important**
> Please note that some of the parameters change in meaning a bit for this type of calculation. In particular, dime should be interpreted as the number of grid points *per processor*. This interpretation helps manage the amount of memory per-processor -- generally the limiting resource for most calculations.

The keywords for this command (described in more detail in the Keywords section) are listed below. All keywords are required (no default values!) unless otherwise noted: **dime** , **ofrac** , **pdime** , **async** , **cglen** , **fglen** , **cgcent** , **fgcent** , **mol** , **lpbe** or **npbe** or **smpbe** , **bcfl** , **ion** (optional) , **pdie** , **sdie** , **chgm** , **usemap** (optional) , **sdens** , **srfm** , **srad** , **swin** , **temp** , **calcenergy** , **calcforce** , **write** , **writemat**

4.2.2.3. Manual multigrid calculation (mg-manual)

This is the standard single-point PBE calculation performed by most solvers. The **mg-manual** calculation offers the most control of parameters to the user. Several of these calculations can be strung together to perform focusing calculations by judicious choice of the **bcfl** flag, however, the setup of the focusing is not automated as it is in **mg-auto** and **mg-para** calculations and therefore this command should only be used by more experienced users.

The keywords for this command (described in more detail in the Keywords section) are listed below. All keywords are required (no default values!) unless otherwise noted: **dime** , **nlev** , **glen** or **grid** , **gcent** , **mol** , **lpbe** or **npbe** or **smpbe** , **bcfl** , **ion** (optional) , **pdie** , **sdie** , **chgm** , **usemap** (optional) , **sdens** , **srfm** , **srad** , **swin** , **temp** , **calcenergy** , **calcforce** , **write** , **writemat**

4.2.2.4. Manual adaptive finite element calculation (fe-manual)

This is a single-point PBE calculation performed by our adaptive finite element PBE solver. It requires that APBS was linked to the Holst group FEtk finite element library (http://www.fetk.org) during compilation.

The finite element solver uses a "solve-estimate-refine" cycle. Specifically, starting from an initial mesh, it performs the following iteration:

1. solve the problem
2. estimate the error in the solution
3. adaptively refine the mesh

until a global error tolerance is reached.

> **Note**
> These methods are most useful for a select set of problems which can benefit from adaptive refinement of the solution. Furthermore, this implementation is experimental. In general, the sequential and parallel focusing multigrid methods offer the most efficient solution of the PBE for most systems

The keywords for this command (described in more detail in the Keywords section) are listed below. All keywords are required (no default values!) unless otherwise noted: **glen** , **etol** , **ekey** , **akeyPRE** , **akeySOLVE** , **targetNum** , **targetRes** , **maxsolve** , **maxvert** , **mol** , **lpbe** or **npbe** or **lrpbe** or **nrpbe** , **bcfl** , **ion** (optional) , **pdie** , **sdie** , **chgm** , **usemap** (optional) , **sdens** , **srfm** , **srad** , **swin** , **temp** , **calcenergy** , **calcforce** , **write** , **writemat**

4.2.2.5. Manual non-numerical calculations (mg-dummy)

This allows users to write out dielectric, ion-accessibility, and charge distribution maps based on biomolecular geometry without actually solving the PB equation. The syntax is identical to mg-dummy.

The keywords for this command (described in more detail in the Keywords section) are listed below. All keywords are required (no default values!) unless otherwise noted: **dime** , **nlev** , **cglen**

or **grid** , **gcent** , **mol** , **lpbe** or **npbe** or **smpbe** , **bcfl** , **ion** (optional) , **pdie** , **sdie** , **chgm** , **usemap** (optional) , **sdens** , **srfm** , **srad** , **swin** , **temp** , **calcenergy** , **calcforce** , **write** , **writemat**

4.2.2.6. Keyword descriptions

This is a list of keywords used in the **ELEC** statements of APBS. Note that not all keywords are used in every **ELEC** statement; see above.

- `akeyPRE` {*key*}

  Specify how the initial finite element mesh should be constructed (from refinement of a very coarse 8-tetrahedron mesh prior to the solve-estimate-refine iteration. This allows for various *a priori* refinement schemes.

  key

  The method used to guide initial refinement:

  unif

  Uniform refinement

  geom

  Geometry-based refinement at molecular surfaces and charges

- `akeySOLVE` {*key*}

  Specify how the the finite element mesh should be adaptively subdivided during the solve-estimate-refine iterations. This allows for various *a posteriori* refinement schemes.

  key

  The method used to guide adpative refinement:

  resi

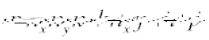
  Residual-based *a posteriori* refinement

- `async` { *rank* }

  This optional keyword allows users to perform the different tasks in a parallel run asynchronously. Specifically, a processor masquerades as process *rank* in a parallel focusing run and provides output (data files and energies/forces) appropriate to that processor's local partition. The user must then assemble the results after all processes complete. First, this option is useful for scheduling on-demand resources: this makes it easy for users to backfill into the available processes in a queue. Second, this option is useful for running on limited resources: this enables users without access to large parallel machines to still perform the same calculations.

  rank

  The ID of the particular processor to masquerade as. Processor IDs range from 0 to *N*-1, where *N* is the total number of processors in the run (see **pdime**). Processor ranks are related to their position in the overall grid by

  where n_x is the number of processors in the x-direction, n_y is the number of processors in the y-direction, n_z is the number of processors in the z-direction, i is the index of the processor in the x-direction, j is the index of the processor in the y-direction, k is the index of the processor in the z-direction, and p is the overall rank of the processor.

- `bcfl` {*flag*}

  Specify the type of boundary conditions used to solve the Poisson-Boltzmann equation:

  flag

  The flag specifying the boundary condition definition:

  zero

  "Zero" boundary condition. Potential at boundary is set to zero. This condition is not commonly used and can result in large errors if used inappropriately.

  sdh

"Single Debye-Hückel" boundary condition. Potential at boundary is set to the values prescribed by a Debye-Hückel model for a single sphere with a point charge, dipole, and quadrupole. The sphere radius is set to the radius of the biomolecule and the sphere charge, dipole, and quadrupole are set to the total moments of the protein. This condition works best when the boundary is sufficiently far from the biomolecule.

mdh

"Multiple Debye-Hückel" boundary condition. Potential at boundary is set to the values prescribed by a Debye-Hückel model for a multiple, non-interacting spheres with a point charges. The sphere radii are set to the atomic radii of the biomolecule and the sphere charges are set to the total charge of the protein. This condition works better than sdh for closer boundaries but can be *very slow* for large biomolecules.

focus

"Focusing" boundary condition. Potential at boundary is set to the values computed by the previous (usually lower-resolution) PB calculation. This is used in sequential focusing performed manually in **mg-manual** calculations. All of the boundary points should lie within the domain of the previous calculation for best accuracy; if any boundary points lie outside, their values are computed using single Debye-Hückel boundary conditions (see above).

- `calcenergy` { *flag* }

This optional keyword controls electrostatic energy output from a PBE calculation.

Note
Note that this option must be used consistently for all calculations that will appear in subsequent **PRINT** statements. For example, if the statement `print energy 1 - 2 end` appears in the input file, then both calculations 1 and 2 must have **calcenergy** keywords present with the same values for *flag*.

flag

Specify the types of energy values to be returned:

no

(Deprecated) don't calculate any energies.

total

Calculate and return total electrostatic energy for the entire molecule.

comps

Calculate and return total electrostatic energy for the entire molecule as well as electrostatic energy components for each atom.

- `calcforce` { *flag* }

This optional keyword controls electrostatic and apolar force output from a PBE calculation.

Note
Note that this option must be used consistently for all calculations that will appear in subsequent **PRINT** statements. For example, if the statement `print force 1 - 2 end` appears in the input file, then both calculations 1 and 2 must have **calcforce** keywords present with the same values for *flag*.

flag

Specify the types of force values to be returned:

no

(Deprecated) don't calculate any forces.

total

Calculate and return total electrostatic and apolar forces for the entire molecule.

comps

Calculate and return total electrostatic and apolar forces for the entire molecule as well as force components for each atom.

- `cgcent` { mol *id* | *xcent ycent zcent* }

  Specify the center of the coarse grid (in a focusing calculation) based on a molecule's center or absolute coordinates. The arguments for this keyword are:

  mol *id*

  Center the grid on molecule with ID *id*; as assigned in the **READ** section. Molecule IDs are assigned in the order they're read, starting at 1.

  xcent ycent zcent

  The coordinates (in Å) at which the grid is centered. Based on the PDB coordinate frame.

- `cglen` {*xlen ylen zlen*}

  Specify the coarse mesh domain lengths in a focusing calculation; this may be different in each direction. This is the starting mesh, so it should be large enough to complete enclose the biomolecule *and* ensure that the chosen boundary condition (see **bcfl**) is appropriate.

  xlen ylen zlen

  Grid lengths in the x-, y-, and z-directions in Å.

- `chgm` {*flag*}

  Specify the method by which the biomolecular point charges (i.e., Dirac delta functions) are mapped onto the grid. As we are attempting to model delta functions, the support (domain) of these discretized charge distributions is always a function of the grid spacing.

  flag

  Discretization method (options have multiple declarations for backward-compatibility):

  spl0

  Traditional trilinear interpolation (linear splines). The charge is mapped onto the nearest-neighbor grid points. Resulting potentials are very sensitive to grid spacing, length, and position.

  spl2

  Cubic B-spline discretization. The charge is mapped onto the nearest- and next-nearest-neighbor grid points. Resulting potentials are somewhat less sensitive (than spl0) to grid spacing, length, and position.

  spl4

  Quintic B-spline discretization. Similar to spl2, except the charge/multipole is additionally mapped to include next-next-nearest neighbors (125 grid points receive charge density).

- `dime` {*nx ny nz*}

  Number of grid points *per processor* for grid-based discretization. For **mg-manual**, the arguments are dependent on the choice of **nlev** by the formula

  $$n = c2^{l+1} + 1$$

  where `n` is the **dime** argument, `c` is a non-zero integer, `l` is the **nlev** value. The most common values for grid dimensions are 65, 97, 129, and 161 (they can be different in each direction); these are all compatible with a **nlev** value of 4. If you happen to pick a "bad" value for the dimensions (i.e., mismatch with **nlev**), the code will adjust the specified dime downwards to more appropriate values. This means that "bad" values will typically result in lower resolution/accuracy calculations! The arguments for this keyword are:

  nx ny nz

  Number of grid points in the x-, y-, and z-directions.

  Important
  Please note that some of the parameters change in meaning a bit for this mg-para calculations. In particular, dime should be interpreted as the number of grid points *per processor*. This interpretation helps manage the

amount of memory per-processor -- generally the limiting resource for most calculations.

- `ekey` { *flag* }

  Specify the method used to determine the error tolerance in the solve-estimate-refine iterations of the finite element solver.

  flag

  Tolerance is interpreted as...

  simp

  ...per-simplex error limit

  global

  ...global (whole domain) error limit

  frac

  ...fraction of simplices you'd like to see refined

- `etol` { *tol* }

  Specify the tolerance for error-based adaptive refinement during the solve-estimate-refine iterations of the finite element solver. See also: **ekey**.

  tol

  The error tolerance for adaptive finite element refinement. The exact definition of this tolerance is determined by the value of **ekey**.

- `fgcent` { mol *id* | *xcent ycent zcent* }

  Specify the center of the fine grid (in a focusing calculation) based on a molecule's center or absolute coordinates. The arguments for this keyword are:

  mol *id*

  Center the grid on molecule with ID *id*; as assigned in the **READ** section. Molecule IDs are assigned in the order they're read, starting at 1.

  xcent ycent zcent

  The coordinates (in Å) at which the grid is centered. Based on the PDB coordinate frame.

- `fglen` {*xlen ylen zlen*}

  Specify the fine mesh domain lengths in a focusing calculation; this may be different in each direction. This should enclose the region of interest in the molecule.

  xlen ylen zlen

  Grid lengths in the x-, y-, and z-directions in Å.

- `gcent` { mol *id* | *xcent ycent zcent* }

  Specify the center of the grid based on a molecule's center or absolute coordinates. The arguments for this keyword are:

  mol *id*

  Center the grid on molecule with ID *id*; as assigned in the **READ** section. Molecule IDs are assigned in the order they're read, starting at 1.

  xcent ycent zcent

  The coordinates (in Å) at which the grid is centered. Based on the PDB coordinate frame.

- `glen` {*xlen ylen zlen*}

  Specify the mesh domain lengths; this may be different in each direction. For some invocations of APBS, either this key or **grid** must be specified.

  xlen ylen zlen

Grid lengths in the x-, y-, and z-directions in Å.

- `grid` {*hx hy hz*}

  Specify the mesh grid spacings; this may be different in each direction. For some invocations of APBS, either this key or **glen** must be specified.

  hx hy hz

  Grid spacings in the x-, y-, and z-directions in Å.

- `ion`
  `charge` {*charge*}
  `charge` {*conc*}
  `radius` {*radius*}

  Specify the mobile ion species present in the system. This command can be repeated as necessary to specify multiple types of ions; however, only the largest ionic radius is used to determine the ion-accessibility function.

  charge

  Mobile ion species charge (in e_c)

  conc

  Mobile ion species concentration (in M)

  radius

  Mobile ion species radius (in Å)

  Warning
  Note that the old command syntax of **ion** {*charge*} {*conc*} {*radius*} is deprecated and will go away "soon".

- `lpbe`

  Specifies that the linearized PBE should be solved. Either this keyword or

  - **npbe**
  - **nrpbe**
  - **lrpbe**
  - **smpbe**

  must be present (based on the calculation type).

- `lrpbe`

  Specifies the linearized form of the regularized PBE equation (RPBE). The regularized PBE equation replaces the point charge distribution with the corresponding Green's function. As a result of this replacement, the solution corresponds to the reaction field instead of the total potential; the total potential can be recovered by adding the appropriate Coulombic terms to the solution. Likewise, this equation immediately yields the solvation energy without the need for reference calculations. Either this keyword or

  - **npbe**
  - **nrpbe**
  - **lrpbe**
  - **smpbe**

  must be present (based on the calculation type).

  Note
  This function is only available for FEM-based solvers.

- `maxsolve` { *num* }

  Specify the number of times to perform the solve-estimate-refine iteration of the finite element solver. See also: **maxvert**, **targetRes**,

  num

  Maximum number of iterations.

- `maxvert` { *num* }

  Specify the maximum number of vertices to allow during solve-estimate-refine cycle of finite

element solver. This places a limit on the memory that can be used by the solver. See also: **targetRes**, **maxsolve**.

num

Maximum number of vertices.

- `mol {id}`

  Specify the molecule for which the PBE is to be solved. IDs are based on the order in which molecules are read by **read mol** statements, starting from 1.

  id

  The ID of the molecule for which the PBE is to be solved.

- `nlev {lev}`

  Specify the depth of the multilevel hierarchy used in the multigrid solver. See **dime** for a discussion of how **nlev** relates to grid dimensions.

  lev

  Depth of the multigrid hierarchy.

- `npbe`

  Specifies that the nonlinear (full) PBE should be solved. Either this keyword or

  - **npbe**
  - **nrpbe**
  - **lrpbe**
  - **smpbe**

  must be present (based on the calculation type).

- `nrpbe`

  Specifies the nonlinear form of the regularized PBE equation (RPBE). The regularized PBE equation replaces the point charge distribution with the corresponding Green's function. As a result of this replacement, the solution corresponds to the reaction field instead of the total potential; the total potential can be recovered by adding the appropriate Coulombic terms to the solution. Likewise, this equation immediately yields the solvation energy without the need for reference calculations. Either this keyword or

  - **npbe**
  - **nrpbe**
  - **lrpbe**
  - **smpbe**

  must be present (based on the calculation type).

  Note
  This function is only available for FEM-based solvers.

- `pdie {diel}`

  Specify the dielectric constant of the biomolecule. This is usually a value between 2 to 20, where lower values consider only electronic polarization and higher values consider additional polarization due to intramolecular motion.

  diel

  Biomolecular dielectric constant (unitless)

- `pdime {npx npy npz}`

  Specify the processor array to be used in a parallel focusing calculation. The product *npx* × *npy* × *npz* should be less than or equal to the total number of processors with which APBS was invoked (usually via **mpirun**). If more processors are provided at invocation than actually used during the run, the extra processors are not used in the calculation. The processors are tiled across the domain in a Cartesian fashion with a specified amount of overlap (see **ofrac**) between each processor to ensure continuity of the solution. Each processor's subdomain will contain the number of grid points specified by the dime keyword.

  npx npy npz

  The number of processors to be used in the x-, y- and z-directions of the system.

- `ofrac` {*frac*}

  Specify the amount of overlap to include between the individual processors meshes in a parallel focusing calculation (see **ofrac**). This should be a value between 0 and 1.

  frac

  Amount of overlap between processor meshes; a value between 0 and 1.

  Tip
  Empirical evidence suggests that an **ofrac** value of 0.1 is sufficient to generate stable energies. However, this value may not be sufficient to generate stable forces and/or good quality isocontours. For example, the following table illustrates the change in energies and visual artifacts in isocontours as a function of **ofrac** values for a small peptide (2PHK:B).

  Table 4.1. Sensitivity of 2PHK:B solvation energy calculations to ofrac values.

ofrac value	Energy (kJ/mol)	Visual artifact in ± 1 kT/e iscontour?
0.05	342.79	No
0.06	342.00	No
0.07	341.12	Yes
0.08	341.14	Yes
0.09	342.02	Yes
0.10	340.84	Yes
0.11	339.67	No
0.12	341.10	No
0.13	341.10	No
0.14	341.32	No
0.15	341.54	No

- `sdie` {*diel*}

  Specify the dielectric constant of the solvent. Bulk water at biologically-relevant temperatures is usually modeled with a dielectric constant of 78-80.

  diel

  Solvent dielectric constant (unitless)

- `sdens` {*density*}

  Specify the number of grid points per square-angstrom to use in surface constructions (e.g., molecular surface, solvent-accessible surface, etc.). Ignored when srad is 0.0 (see **srad**) or srfm is spl2 (see **srfm**). There is a direct correlation between this value used for the surface sphere density, the accuracy of the surface calculations,, and the APBS calculation time. APBS "suggested" value is 10.0.

  density

  Surface sphere density (in grid points/Å^2).

- `smpbe`
  `vol` { *volume* }
  `size` { *number* }

  Specifies that the size-modified PBE should be solved as described by Chu V, et al *Biophys J*, in press (doi:10.1529/biophysj.106.099168). The parameter *radius* controls the lattice size (in Angstroms) used in the SMPBE formalism; each lattice site has a volume equal to *radius*^{3}. The parameter *size* controls the relative size of the ions (in Angstroms) such that each lattice site can contain a single ion of volume *radius*^{3} or *size* ions of volume *radius*^{3}/*size*. Either this keyword or

  - **npbe**
  - **nrpbe**
  - **lrpbe**
  - **smpbe**

  must be present (based on the calculation type).

- `srad` {*radius*}

  Specify the radius of the solvent molecules; this parameter is used to define the dielectric function (see **srfm**). This value is usually set to 1.4 Å for water.

  radius

  Solvent radius (in Å).

- `swin` {*win*}

  Specify the size of the support (i.e., the rate of change) for spline-based surface definitions (see **srfm**). Usually 0.3 Å.

  win

  Spline window (in Å).

- `srfm` {*flag*}

  Specify the model used to construct the dielectric ion-accessibility coefficients.

  flag

  The coefficient model:

  mol

  The dielectric coefficient $\varepsilon(x)$ is defined based on a molecular surface definition. The problem domain is divided into two spaces. The "free volume" space is defined by the union of solvent-sized spheres (see **srad**) which do not overlap with biomolecular atoms. This free volume is assigned bulk solvent dielectric values. The complement of this space is assigned biomolecular dielectric values. With a non-zero solvent radius (**srad**), this choice of coefficient corresponds to the traditional definition used for PB calculations. When the solvent radius is set to zero, this corresponds to a van der Waals surface definition.

  The ion-accessibility coefficient $\bar{\varepsilon}(x)$ is defined by an "inflated" van der Waals model. Specifically, the radius of each biomolecular atom is increased by the radius of the ion species. The problem domain is then divided into two spaces. The space inside the union of these inflated atomic spheres is assigned an ion-accessibility value of 0; the complement space is assigned bulk ion accessibility values.

  smol

  The dielectric and ion-accessibility coefficients are defined as for mol (see above). However, they are then "smoothed" by a 9-point harmonic averaging to somewhat reduce sensitivity to the grid setup as described by Bruccoleri et al. *J Comput Chem* 18 268-276, 1997 (doi:10.1002/(SICI)1096-987X(19970130)18:2>268::AID-JCC11>3.0.CO;2-E).

  spl2

  The dielectric $\varepsilon(x)$ and ion-accessibility $\bar{\varepsilon}(x)$ coefficients are defined by a cubic-spline surface as described by Im et al, *Comp Phys Commun* 111 (1-3) 59-75, 1998 (doi:10.1016/S0010-4655(98)00016-2). These spline-based surface definitions are very stable with respect to grid parameters and therefore ideal for calculating forces. However, they require substantial reparameterization of the force field; interested users should consult Nina et al, *Biophys Chem* 78 (1-2) 89-96, 1999 (doi:10.1016/S0301-4622(98)00236-1). Additionally, these surfaces can generate unphysical results with non-zero ionic strengths; this is an on-going area of development.

  spl4

  The dielectric and ion-accessibility coefficients are defined by a 7th order polynomial. This surface definition has characteristics similar to spl2, but provides higher order continuity necessary for stable force calculations with atomic multipole force fields (up to quadrupole).

- `targetRes` { *res* }

  Specify the target resolution of the simplices in the mesh; refinement will continue until the longest edge of every simplex is below this value. See also: **maxvert**, **maxsolve**, **targetNum**

res

Target resolution for longest edges of simplices in mesh (in Å).

- `targetNum` { *num* }

Specify the target number of vertices in the *initial* mesh; intiial refinement will continue until this number is reached or the the longest edge of every simplex is below **targetNum**. See also: **targetRes**

num

Target number of vertices in initial mesh.

- `temp` { *T* }

Temperature for PBE calculation.

T

Temperature (in K)

- `usemap` {*type*} {*id*}

Specify pre-calculated coefficient maps to be used in the PB calculation. These must have been input via an earlier **read map** statement.

type

Specify the type of pre-calculated map to be read in:

diel

Dielectric function map (as read by **read map** *dielx diely dielx*); this causes the **pdie**, **sdie**, **srad**, **swin**, and **srfm** parameters and the radii of the biomolecular atoms to be ignored when computing dielectric values for the PBE.

kappa

Mobile ion-accessibility function map (as read by **read map** *kappa*); this causes the **swin** and **srfm** parameters and the radii of the biomolecular atoms to be ignored when computing mobile ion values for the PBE. The **ion** parameter is not ignored and will still be used.

charge

Charge distribution map (as read by **read map** *charge*); this causes the **chgm** parameter and the charges of the biomolecular atoms to be ignored when assembling the fixed charge distribution for the PBE.

id

This ID specifies the particular map read in with **read map**. These IDs are assigned sequentially, starting from 1, for each map type read by APBS.

- `write` {type} {format} {stem}

This controls the output of scalar data calculated during the PB run. This keyword can be repeated several times to provide various types of data.

type

What type of data to output:

charge

Write out the biomolecular charge distribution in units of e_c (electron charge) per Angstrom^3 (multigrid only).

pot

Write out the electrostatic potential in units of k_bT/e_c (multigrid and finite element)

smol

Write out the solvent accessibility defined by the molecular surface definition (see **srfm** smol). Values are unitless and range from 0 (inaccessible) to 1 (accessible). (multigrid and finite element)

sspl

Write out the spline-based solvent accessibility (see **srfm** spl2). Values are unitless and range from 0 (inaccessible) to 1 (accessible). (multigrid and finite element)

vdw

Write out the van der Waals-based solvent accessibility (see **srfm** smol with **srad** smol). Values are unitless and range from 0 (inaccessible) to 1 (accessible). (multigrid and finite element)

ivdw

Write out the inflated van der Waals-based ion accessibility (see **srfm** smol). Values are unitless and range from 0 (inaccessible) to 1 (accessible). (multigrid and finite element)

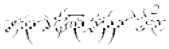
lap

Write out the Laplacian of the potential

in units of $k_BT/e_c/Å^2$ (multigrid only).

edens

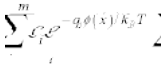
Write out the "energy density"



in units of $k_BT/e_c/Å^2$ (multigrid only).

ndens

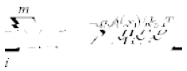
Write out the mobile ion number density



for m ion species in units of M (multigrid only).

qdens

Write out the mobile charge density



for m ion species in units of e_c M (multigrid only).

dielx

Write out the dielectric map shifted by 1/2 grid spacing in the x-direction (see **read** diel). The values are unitless (multigrid only).

diely

Write out the dielectric map shifted by 1/2 grid spacing in the y-direction (see **read** diel). The values are unitless (multigrid only).

dielz

Write out the dielectric map shifted by 1/2 grid spacing in the z-direction (see **read** diel). The values are unitless (multigrid only).

kappa

Write out the ion-accessibility kappa map (see **read** kappa). The values are in units of $Å^{-2}$ (multigrid only).

format

Specify the format for writing out the data.

dx

Write out data in OpenDX format. This is the preferred format for APBS I/O. (multigrid and finite element).

avs

Write out data in AVS UCD format. (finite element only)

uhbd

Write out data in UHBD format. (multigrid only)

stem

Specify the path for the output; files are written to *stem*.XXX, where XXX is determined by the file format (and processor rank for parallel calculations).

- writemat {type} {stem}

  This controls the output of the mathematical operators in the PBE as matrices in Harwell-Boeing format (multigrid only)

  type

  What type of operator to output:

  poission

  Write out the Poisson operator.

  pot

  Write out the Gateaux (functional) derivative of the full PBE operator evaluated at the current solution.

  stem

  Specify the path for the output; files are written to *stem*.mat.

4.2.3. APOLAR statements

APOLAR [name *id*] [*keywords*...]
END

This section is the main component for apolar solvation calculations in APBS runs. There may be several **APOLAR** sections, operating on different molecules or using different parameters for multiple runs on the same molecule. The order of the **APOLAR** statement matters (see above); the arguments are:

name *id*

This optional command allows users to assign an alphanumeric string to the calculation to facilitate later operations (particularly in the **PRINT** statements).

keywords

Keywords describing the parameters of the apolar calculation. These are described in Section 4.2.3.2, "Keyword descriptions".

4.2.3.1. Basic apolar solvation calculations

APBS apolar calculations follow the very generic framework described in:

Wagoner JA, Baker NA. Assessing implicit models for nonpolar mean solvation forces: the importance of dispersion and volume terms. Proc Natl Acad Sci USA, 103, 8331-8336, 2006. (http://dx.doi.org/10.1073/pnas.0600118103)

In particular, nonpolar solvation potentials of mean force (energies) are calculated according to Equation 4.1, "Nonpolar solvation potentials of mean force".

Equation 4.1. Nonpolar solvation potentials of mean force

$$W^{(\mathrm{np})}(\mathbf{x}) = \gamma A(\mathbf{x}) + pV(\mathbf{x}) + \overline{\rho} \int_{\Omega} u^{(\mathrm{att})}(\mathbf{x},\mathbf{y})\theta(\mathbf{y})d\mathbf{y}$$

Mean nonpolar solvation forces are calculated according to Equation 4.2, "Nonpolar solvation mean forces".

Equation 4.2. Nonpolar solvation mean forces

$$\mathbf{F}_i^{(\mathrm{np})}(\mathbf{x}) = -\gamma \frac{\partial A(\mathbf{x})}{\partial \mathbf{x}_i} - p \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}_i} - \bar{\rho} \int_\Omega \frac{\partial u^{(\mathrm{att})}(\mathbf{x}, \mathbf{y})}{\partial \mathbf{x}_i} \theta(\mathbf{y}) d\mathbf{y}$$

In these equations, gamma (see gamma below) is the repulsive (hard sphere) solvent surface tension, A is the conformation-dependent solute surface area (see srad and srfm below), p (see press below) is the repulsive (hard sphere) solvent pressure, V is the conformation-dependent solute volume (see srad and srfm below), rho (see bconc below) is the bulk solvent density, and the integral involves the attractive portion (defined in a Weeks-Chandler-Andersen sense) of the Lennard-Jones interactions between the solute and the solvent integrated over the region of the problem domain *outside* the solute volume V (see srad and srfm below). Lennard-Jones parameters are taken from APBS parameter files (see Section A.2.2, "XML Parameter Format ") as read in through an APBS input file READ statement.

> **Important**
> All apolar calculations require a parameter file which contains Lennard-Jones radius and well-depth parameters for all the atoms in the solute PDB. This parameter file must also contain radius and well-depth parameters for water (specifically: residue "WAT" and atom "OW").

Note that the above expressions can easily be reduced to simpler apolar solvation formalisms by setting one or more of the coefficients (gamma, rho, and p) to zero through the keywords below.

4.2.3.2. Keyword descriptions

This is a list of keywords used in the **APOLAR** statements of APBS.

- `bconc` {*bulk solvent density*}

  Specify the bulk solvent density in Å^{-3}. This coefficient multiplies the integral term of the apolar model discussed above and can be set to zero to eliminate integral contributions to the apolar solvation calculation.

  bulk solvent density

  Bulk solvent density (in Å^{-3})

- `calcenergy` { *flag* }

  This optional keyword controls electrostatic energy output from an apolar solvation calculation.

  > **Note**
  > Note that this option must be used consistently for all calculations that will appear in subsequent **PRINT** statements. For example, if the statement `print energy 1 - 2 end` appears in the input file, then both calculations 1 and 2 must have **calcenergy** keywords present with the same values for *flag*.

  flag

  Specify the types of energy values to be returned:

  no

  (Deprecated) don't calculate any energies.

  total

  Calculate and return total apolar energy for the entire molecule.

  comps

  Calculate and return total apolar energy for the entire molecule as well as electrostatic energy components for each atom.

- `calcforce` { *flag* }

  This optional keyword controls apolar force output.

Note
Note that this option must be used consistently for all calculations that will appear in subsequent **PRINT** statements. For example, if the statement `print force 1 - 2 end` appears in the input file, then both calculations 1 and 2 must have **calcforce** keywords present with the same values for *flag*.

flag

Specify the types of force values to be returned:

no

(Deprecated) don't calculate any forces.

total

Calculate and return total apolar forces for the entire molecule.

comps

Calculate and return total apolar forces for the entire molecule as well as force components for each atom.

- `dpos` {*displacement*}

  This is the displacement used for finite-difference-based calculations of surface area derivatives. I know: this is a terrible way to calculate surface area derivatives -- we're working on replacing it with an analytic version. In the meantime, please use this parameter with caution.

  Important
  This parameter is very dependent on sdens; e.g., smaller values of `dpos` require larger values of sdens.

  dispalcement

  Finite difference displacement for force (surface area derivative) calculations in units of Å.

- `gamma` { *value* }

  The coefficient (surface tension) for solvent-accesisble surface area (SASA) models of apolar solvation. This term can be set to zero to eliminate SASA contributions to the apolar solvation calculations.

  value

  SASA-based apolar coefficient (in kJ/mol/Å).

- `grid` {*hx hy hz*}

  Specify the quadrature grid spacing for volume integral calculations.

  hx hy hz

  Quadrature spacing in the x-, y-, and z-directions in Å.

- `mol` {*id*}

  Specify the molecule for which the apolar calculation is to be performed. IDs are based on the order in which molecules are read by **read mol** statements, starting from 1.

  id

  The ID of the molecule for which the apolar calculation is to be performed.

- `press` {*solvent pressure*}

  Specify the solvent pressure in kJ mol^{-1} $Å^{-3}$. This coefficient multiplies the volume term of the apolar model discussed above and can be set to zero to eliminate volume contributions to the apolar solvation calculation.

  solvent pressure

  Solvent pressure (in kJ mol^{-1} $Å^{-3}$)

- `sdens` {*surface discretization density*}

Specify the number of grid points per square-angstrom to use in surface calculations (e.g., molecular surface, solvent accesible surface). Ignored when srad is 0.0 (see **srad**) or srfm is spl2 (see **srfm**). Users beware: there is a direct correlation between the value used for the sphere density, the accuracy of the results, and the APBS calculation time.

surface discretization density

Surface sphere density (in grid points/Å^2).

- `srad` {*radius*}

  Specify the radius of the solvent molecules; this parameter is used to define various solvent-related surfaces and volumes (see **srfm**). This value is usually set to 1.4 Å for water.

  radius

  Solvent radius (in Å).

- `swin` {*win*}

  Specify the size of the support (i.e., the rate of change) for spline-based surface definitions (see **srfm**). Usually 0.3 Å.

  win

  Spline window (in Å).

- `srfm` {*flag*}

  Specify the model used to construct the solvent-related surface and volume

  Note
  Under construction: we're in the process of adding additional surface definitions.

  flag

  The surface/volume model:

  sacc

  Solvent-accessible (also called "probe-inflated") surface and volume.

- `temp` { *T* }

  Temperature for calculation.

  T

  Temperature (in K)

4.2.4. PRINT statements

```
PRINT {what} [id op id op...]
END
```

This is a very simple section that allows linear combinations of calculated properties to be written to standard output. It has the following variables:

what

Specify which quantities to manipulate/print:

energy

Print energies as calculated with an earlier **calcenergy** **ELEC** command.

Warning
This usage is deprecated and will be replaced in the next release. Please use "elecEnergy" or "apolEnergy" as appropriate. For now, this will return the old results of "elecEnergy".

force

Print forces as calculated with an earlier **calcforce** **ELEC** command.

Warning
This usage is deprecated and will be replaced in the next release.

Please use "elecForce" or "apolForce" as appropriate. For now, this will return the old results of "elecEnergy".

elecEnergy

Print energies as calculated with an earlier **calcenergy** **ELEC** command.

elecForce

Print forces as calculated with an earlier **calcforce** **ELEC** command.

apolEnergy

Print energies as calculated with an earlier **calcenergy** **APOLAR** command.

apolForce

Print forces as calculated with an earlier **calcforce** **APOLAR** command.

id

The ID of a particular **ELEC** calculation as specified with the **ELEC name** *id* command. If the *id* variables are not set explicitly, they are assigned sequential integers, starting at 1, based on the order of the **ELEC** statements.

op

Specify the arthimetic operation to be performed on the calculated quantities:

+

Addition

-

Subtraction

Given all these options, a typical declaration might look like:

Example 4.2. PRINT statement example

```
# Energy change due to binding
print energy complex - ligand - protein end
# Energy change due to solvation
print energy solvated - reference end
# Solvation energy change due to binding
print energy
complex_solv - complex_ref
- ligand_solv + ligand_ref
- protein_solv + protein_ref
end
```

See the APBS `examples/` directory for more examples.

4.3. Using APBS with other programs

APBS was designed to facilitate use with other programs. This section outlines some of the programs with which APBS is known to work. However, it is likely that applications which use APBS have been inadvertantly omitted from this list. If you know of software that uses APBS and is not listed here, please contact Nathan Baker <baker@biochem.wustl.edu>.

4.3.1. Web interfaces

4.3.1.1. Gemstone

The Gemstone extension (http://gemstone.mozdev.org/) for the Firefox web browser provides a very easy-to-use interface to APBS with all of the functionality of the command-line interface. This extension currently uses web services provided by NBCR).

The NBCR Opal Web Services client for APBS (http://nbcr.net/services/#Software) provides a more basic interface which allows users to execute APBS jobs remotely via Python or build such functionality into their own software.

4.3.2. Graphical user interfaces

4.3.2.1. PyMOL

PyMOL (http://pymol.sourceforge.net/) is a molecular visualization and animation package which provides an interface to APBS. The APBS plugin to PyMOL (developed by Michael George Lerner) permits isocontour and surface map visualization of APBS results.

4.3.2.2. VMD

VMD (http://www.ks.uiuc.edu/Research/vmd/) is a molecular visualization and animation package which provides an interface to APBS. It permits visualization of APBS results as isocontours, electric field lines, or on biomolecular surfaces. VMD also a graphical plugin to setup APBS calculations and execute them either locally or remotely via BioCoRE. More information is available at http://www.ks.uiuc.edu/Research/vmd/plugins/apbsrun/.

4.3.2.3. PMV

PMV (http://www.scripps.edu/~sanner/python) is a Python-based molecular visualization package which provides an interface to APBS. It not only permits visualization of APBS results but it also integrates setup and executation of APBS calculations. The PMV/APBS interface (http://mccammon.ucsd.edu/pmv_apbs/) is under active development and future versions will offer even more setup, visualization, and analysis functionality.

The APBS interface is distributed with recent beta versions of PMV, avalaible from http://www.scripps.edu/~sanner/python. Additional documention for using APBS with PMV is provided at http://mccammon.ucsd.edu/~jswanson/apbsDoc/command_doc2.html.

4.3.3. Simulation software

Robert Konecny (McCammon Group) has developed iAPBS (http://mccammon.ucsd.edu/iapbs/), an interface between APBS and the simulation packages AMBER, CHARMM, and NAMD. More information is available from the iAPBS homepage: http://mccammon.ucsd.edu/iapbs/ .

4.3.4. Visualization software

Electrostatic potentials are commonly visualized in the context of biomolecular structure to better understand functional aspects of biological systems. This section describes molecular graphics software which can display potentials and other data output from APBS. Note that this set of programs also includes the graphical users interfaces listed above.

4.3.4.1. Dino3D

Dino3D (http://www.dino3d.org/ is a molecular graphics program which can read UHBD-format electrostatic data. APBS can write multigrid results in UHBD format (see the **write** **ELEC** command) and therefore can be used with Dino3D.

4.3.4.2. MOLMOL

MOLMOL (http://www.mol.biol.ethz.ch/wuthrich/software/molmol/) is a molecular graphics package with an emphasis on NMR-generated structural data. A program is provided with APBS (see `tools/mesh` and the Data conversion tools in this manual) which converts OpenDX data to MOLMOL format.

4.3.4.3. OpenDX

OpenDX (http://www.opendx.org) is a general data visualization package which can read APBS output using the scripts provided in `tools/visualization/opendx` (see the discussion of Data visualization tools in this manual). However, as there is no straightforward way to visualizate the potential in the context of the atomic structure, OpenDX should not a first choice for APBS visualization.

Chapter 5. Getting help

Table of Contents

APBS follows an open source software development model and relies heavily on the user community for feedback to enhance the software, identify bugs, etc.

5.1. User Forums

The SourceForge project page offers many user forums for discussion on APBS topics. Users are

encouraged to post bugs and request for support and features via the following forums:

- Bugs - A forum for listing and displaying progress on APBS bugs.
- Support Requests - A forum for requesting support for installation on various architectures and general APBS usage.
- Feature Requests - A forum for feature requests and other desired additions to APBS.

5.2. Mailing Lists

There are additionally two mailing lists for the APBS software:

Announcements

This is a very low-traffic list to inform users of new APBS releases and APBS-related software (PDB2PQR, etc.). news (updates, bugs, etc.) on this low-traffic list.

- List information: http://lists.sourceforge.net/lists/listinfo/apbs-announce
- List archives: http://sourceforge.net/mailarchive/forum.php?forum=apbs-announce

User discussion

This higher-traffic list is the primary forum for discussion of APBS functionality, usage, possible enhancements, and bugs.

- List information: http://lists.sourceforge.net/lists/listinfo/apbs-users
- List archives: http://sourceforge.net/mailarchive/forum.php?forum=apbs-users

5.3. Bug and other problems

Bugs or potential bugs, problems, etc. should either be posted to the SourceForge bug tracker forum or reported to the apbs-users (http://lists.sourceforge.net/lists/listinfo/apbs-users) mailing list. Please see the user forums or mailing list sections of the documentation for more information.

Chapter 6. Programming

Users who are interested in developing code for or with APBS should consult the Programmer's Guide, available in HTML format at `doc/html/programmer/index.html` .

Appendix A. File formats

Table of Contents

This section introduces some of the input and output file formats which are used by APBS.

A.1. PQR biomolecular structure format

The PQR format is the primary input format for biomolecular structure in APBS package.

A.1.1. PQR flat-file format

This format is a modification of the PDB format which allows users to add charge and radius parameters to existing PDB data while keeping it in a format amenable to visualization with standard molecular graphics programs. The origins of the PQR format are somewhat uncretain, but has been used by several computational biology software programs, including MEAD and

AutoDock. UHBD uses a very similar format called QCD.

APBS reads very loosely-formatted PQR files: all fields are whitespace-delimited, thereby allowing coordinates which are larger/smaller than ± 999 Å.

APBS reads data on a per-line basis from PQR files using the following format:

`Field_name` `Atom_number` `Atom_name` `Residue_name` *`Chain_ID`* `Residue_number` `X` `Y` `Z` `Charge` `Radius`

where the whitespace is the most important feature of this format. The fields are:

`Field_name`

A string which specifies the type of PQR entry and should either be `ATOM` or `HETATM` in order to be parsed by APBS.

`Atom_number`

An integer which provides the atom index.

`Atom_name`

A string which provides the atom name.

`Residue_name`

A string which provides the residue name.

`Chain_ID`

An optional string which provides the chain ID of the atom.

Note
NOTE: Chain ID support is a new feature of APBS 0.5.0 and later versions.

`Residue_number`

An integer which provides the residue index.

`X Y Z`

3 floats which provide the atomic coordiantes.

`Charge`

A float which provides the atomic charge (in electrons).

`Radius`

A float which provides the atomic radius (in Å).

Clearly, this format can deviate wildly from PDB, particularly when large coordinate values are used. However, in order to maintain compatibility with most molecular graphics programs, the PDB2PQR utilities provided with apbs (see the Parameterization section) attempt to preserve the PDB format as much as possible.

A.1.2. PQR XML Format

The PQR XML format was designed to remediate some of the shortcomings of the flat-file format. By use of XML, issues related to extra fields in the file or columns merging together can easily be remedied. Additionally, APBS will only parse the necessary information from the XML file and will ignore all other information, so users wishing to store extra data related to a residue or atom can do so inline without affecting APBS.

This data format has the following form:

```
# Comments
<roottag>
  <residue>
...
  <atom>
    <x>x</x>
    <y>y</y>
    <z>z</z>
    <charge>charge</charge>
    <radius>radius</radius>
  </atom>
...
  </residue>
```

```
...
</roottag>
```

Note that the `residue` tag and its elements are completely optional - APBS only reads in atom data. Fields such as residue name, chain ID, and other residue-specific data will be ignored.

The variables in this example are:

Comments

Any number of comment lines, each line starting with the "#" symbol

roottag

This is the root element of the XML file. The value is not important to APBS - APBS simply checks that it is closed at the end of the file.

x

A float giving the x-coordinate of the atom.

y

A float giving the y-coordinate of the atom.

z

A float giving the z-coordinate of the atom.

charge

A float giving the atomic charge (in electrons).

atomradius

A float giving the atomic Radius (in Å).

A.2. Parameter file format

APBS uses parameter files with the **READ parm** command.

A.2.1. Flat-file format

The parameter file is a series of lines of the format:

Residue_name Atom_name Charge Radius Epsilon

where the whitespaces are important and denote separation between the fields. The fields here are:

Residue_name

A string giving the residue name, as provided in the PDB file to be parameterized.

Atom_name

A string giving the atom name, as provided in the PDB file to be parameterized.

Charge

A float giving the atomic charge (in electrons).

Radius

A float giving the atomic Radius (in Å).

Epsilon

A float giving the Lennard-Jones well depth (in kJ/mol). This is used for the calculation of WCA energies in apolar solvation energies and forces. We assume that the Lennard-Jones potential is defined in the "AMBER style":

$$U_{\mathrm{LJ}}(r) = \epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - 2 \left(\frac{\sigma}{r} \right)^{6} \right]$$

A.2.2. XML Parameter Format

This data format has the following form:

```
# Comments
<ffname>
  <residue>
  <name>resname</name>
  <atom>
    <name>atomname</name>
    <charge>atomcharge</charge>
    <radius>atomradius</radius>
    <epsilon>atomepsilon</epsilon>
  </atom>
...
  </residue>
...
</ffname>
```

The variables in this example are:

Comments

Any number of comment lines, each line starting with the "#" symbol

ffname

The name of the forcefield. This is the root element of the XML file.

resname

A string giving the residue name, as provided in the PDB file to be parameterized.

atomname

A string giving the atom name, as provided in the PDB file to be parameterized.

atomcharge

A float giving the atomic charge (in electrons).

atomradius

A float giving the atomic Radius (in Å).

atomepsilon

A float giving the Lennard-Jones well depth (in kJ/mol). This is used for the calculation of WCA energies in apolar solvation energies and forces. We assume that the Lennard-Jones potential is defined in the "AMBER style":

$$U_{\mathrm{LJ}}(r) = \epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - 2 \left(\frac{\sigma}{r} \right)^{6} \right]$$

A.3. Harwell-Boeing column-compressed matrix format

This is the sparse matrix output format used by APBS for analyses of the matrix operators which are constructed during PB solution. This format was implemented so matrix operators could by decomposed with SuperLU and ARPACK.

Important
Details of the format are complicated; this section is under construction.

A.4. OpenDX scalar data format

We output most discretized scalar data (e.g., potential, accessibility, etc.) from APBS in the data format used by the OpenDX software package (http://www.opendx.org). The OpenDX data format is very flexible; the following sections describe the application of this format for APBS multigrid and finite element datasets.

A.4.1. Multigrid

This data format has the following form:

```
# Comments
object 1 class gridpositions counts nx ny nz
origin xmin ymin zmin
delta hx 0.0 0.0
```

```
delta 0.0 hy 0.0
delta 0.0 0.0 hz
object 2 class gridconnections counts nx ny nz
object 3 class array type double rank 0 times n data follows
u(0,0,0) u(0,0,1) u(0,0,2)
...
u(0,0,nz-3) u(0,0,nz-2) u(0,0,nz-1)
u(0,1,0) u(0,1,1) u(0,1,2)
...
u(0,1,nz-3) u(0,1,nz-2) u(0,1,nz-1)
...
u(0,ny-1,nz-3) u(0,ny-1,nz-2) u(0,ny-1,nz-1)
u(1,0,0) u(1,0,1) u(1,0,2)
...
attribute "dep" string "positions"
object "regular positions regular connections" class field
component "positions" value 1
component "connections" value 2
component "data" value 3
```

The variables in this example are:

Comments

Any number of comment lines, each line starting with the "#" symbol

nx ny nz

The number of grid points in the x-, y-, and z-directions.

xmin ymin zmin

The coordinates of the grid lower corner.

hx hy hz

The grid spacings in the x-, y-, and z-directions.

n

The total number of grid points; *n* = *nx* * *ny* * *nz*

u(,*,*)*

The data values, ordered with the z-index increasing most quickly, followed by the y-index, and then the x-index.

A.4.2. Finite element

For finite element solutions, the OpenDX format takes the following form:

```
object 1 class array type float rank 1 shape 3 items N
v1x v1y v1z
v2x v2y v2z
...
vNx vNy vNz
object 2 class array type int rank 1 shape 4 items M
s1a s1b s1c s1d
s2a s2b s2c s2d
...
sMa sMb sMc sMd
attribute "element type" string "tetrahedra"
object 3 class array type float rank 0 items N
u1
u2
...
uN
attribute "dep" string "positions"
object "irregular positions irregular connections" class field
component "positions" value 1
component "connections" value 2
component "data" value 3
end
```

where the variables are:

N

Number of vertices

vix viy viz

Coordinates of vertex *i*

M

Number of simplices

sia sib sic sid

IDs of vertices in simplex *i*

ui

Data value associated with vertex *i*

Appendix B. License

APBS is covered under the GNU Public license (GPL), which basically means you can copy it, change it, use subsets of it, redistribute it, etc.; *however*, you need to give credit to the original source *and* the original portion of the code must remain under the GPL.

Here is the specific licensing information: