

APBS

1.3

Generated by Doxygen 1.7.4

Fri Jun 29 2012 07:43:50

Contents

Chapter 1

APBS Programmers Guide

APBS was written by Nathan A. Baker.

Additional contributing authors listed in the code documentation.

1.1 Table of Contents

- Programming Style
- Application programming interface documentation
 - Modules
 - Class list
 - Class members
 - Class methods

1.2 License

Primary author: Nathan A. Baker (nathan.baker@pnnl.gov)

Pacific Northwest National Laboratory

Additional contributing authors are listed in the code documentation.

Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific Northwest Division for the U.S. Department of Energy.

Portions Copyright (c) 2002-2010, Washington University in St. Louis.

Portions Copyright (c) 2002-2010, Nathan A. Baker.

Portions Copyright (c) 1999-2002, The Regents of the University of California.

Portions Copyright (c) 1995, Michael Holst.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the developer nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This documentation provides information about the programming interface provided by the APBS software and a general guide to linking to the APBS libraries. Information about installation, configuration, and general usage can be found in the [User's Guide](#).

1.3 Programming Style

APBS was developed following the [Clean OO C](#) style of Mike Holst. In short, Clean OO C code is written in a object-oriented, ISO C-compliant fashion, and can be compiled with either a C or C++ compiler.

Following this formalism, all public data is enclosed in structures which resemble C++ classes. These structures and member functions are then declared in a public header file which provides a concise description of the interface for the class. Private functions and data are included in private header files (or simply the source code files themselves) which are not distributed. When using the library, the end-user only sees the public header file and the compiled library and is therefore (hopefully) oblivious to the private members and functions. Each class is also equipped with a constructor and destructor function which is responsible for allocating and freeing any memory required by the instantiated objects.

As mentioned above, public data members are enclosed in C structures which are visible to the end-user. Public member functions are generated by mangling the class and function names *and* passing a pointer to the object on which the member function is supposed to act. For example, a public member function with the C++ declaration

```
public double Foo::bar(int i, double d)
```

would be declared as

```
VEXTERNC double Foo_bar(Foo *thee, int i, double d)
```

where `VEXTERNC` is a compiler-dependent macro, the underscore `_` replaces the C++ double-colon `::`, and `thee` replaces the `this` variable implicit in all C++ classes. Since they do not appear in public header files, private functions could be declared in any format pleasing to the user, however, the above declaration convention should generally be used for both public and private functions. Within the source code, the public and private function declarations/definitions are prefaced by the macros `VPUBLIC` and `VPRIVATE`, respectively. These are macros which reduce global name pollution, similar to encapsulating private data within C++ classes.

The only C++ functions not explicitly covered by the above declaration scheme are the constructors (used to allocate and initialize class data members) and destructors (used to free allocated memory). These are declared in the following fashion: a constructor with the C++ declaration

```
public void Foo::Foo(int i, double d)
```

would be declared as

```
VEXTERNC Foo* Foo_ctor(int i, double d)
```

which returns a pointer to the newly constructed `Foo` object. Likewise, a destructor declared as

```
public void Foo::~Foo()
```

in C++ would be

```
VEXTERNC void Foo_dtor(Foo **thee)
```

in Clean OO C.

Finally, inline functions in C++ are simply treated as macros in Clean OO C and declared/defined using `define` statements in the public header file.

See any of the APBS header files for more information on Clean OO C programming styles.

1.4 Application programming interface documentation

The API documentation for this code was generated by `doxygen`. You can either view the API documentation by using the links at the top of this page, or the slight re-worded/re-interpreted list below:

- [Class overview](#)
- [Class declarations](#)
- [Class members](#)
- [Class methods](#)

Chapter 2

Todo List

Global `Vfetk_PDE_initElement(PDE *thee, int elementType, int chart, double tvx[][VAPBS_DIM], void *data)`
Jump term is not implemented

Chapter 3

Deprecated List

Global **sMGparm::nlev** Just ignored now

Chapter 4

Bug List

Global `Bmat_printHB(Bmat *thee, char *fname)` Hardwired to only handle the single block symmetric case.

Class `sVpmgp` Value ipcon does not currently allow for preconditioning in PMG

Global `Vacc_fastMolAcc(Vacc *thee, double center[VAPBS_DIM], double radius)`
This routine has a slight bug which can generate very small internal regions of high dielectric (thanks to John Mongan and Jess Swanson for finding this)

Global `Vacc_molAcc(Vacc *thee, double center[VAPBS_DIM], double radius)` This routine has a slight bug which can generate very small internal regions of high dielectric (thanks to John Mongan and Jess Swanson for finding this)

Global `Vfetk_dumpLocalVar()` This function is not thread-safe

Global `Vfetk_externalUpdateFunction(SS **simps, int num)` This function is not thread-safe.

Global `Vfetk_fillArray(Vfetk *thee, Bvec *vec, Vdata_Type type)` Several values of type are not implemented

Global `Vfetk_PDE_ctor(Vfetk *fetk)` Not thread-safe

Global **Vfetk_PDE_ctor2**(PDE *thee, Vfetk *fetk) Not thread-safe

Global **Vfetk_PDE_delta**(PDE *thee, int type, int chart, double txq[], void *user, double F[])
This function is not thread-safe

Global **Vfetk_PDE_DFu_wv**(PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[])
This function is not thread-safe

Global **Vfetk_PDE_Fu**(PDE *thee, int key, double F[]) This function is not thread-safe
This function is not implemented (sets error to zero)

Global **Vfetk_PDE_Fu_v**(PDE *thee, int key, double V[], double dV[][VAPBS_DIM])
This function is not thread-safe

Global **Vfetk_PDE_initElement**(PDE *thee, int elementType, int chart, double tvx[][VAPBS_DIM], void *data)
This function is not thread-safe

Global **Vfetk_PDE_initFace**(PDE *thee, int faceType, int chart, double tnvec[]) This
function is not thread-safe

Global **Vfetk_PDE_initPoint**(PDE *thee, int pointType, int chart, double txq[], double tU[], double tdU[][VAPBS_DIM])
This function is not thread-safe
This function uses pre-defined boundary definitions for the molecular surface.

Global **Vfetk_PDE_Ju**(PDE *thee, int key) This function is not thread-safe.

Global **Vfetk_PDE_markSimplex**(int dim, int dimll, int simplexType, int faceType[VAPBS_NVS], int vertexTy)
This function is not thread-safe

Global **Vfetk_PDE_u_D**(PDE *thee, int type, int chart, double txq[], double F[]) This
function is hard-coded to call only multiple-sphere Debye-Hückel functions.
This function is not thread-safe.

Global `Vfetk_PDE_u_T`(PDE *thee, int type, int chart, double txq[], double F[]) This function is not thread-safe.

Global `Vfetk_write`(Vfetk *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, Bvec *vec,

Some values of format are not implemented

Global `Vgreen_helmholtz`(Vgreen *thee, int npos, double *x, double *y, double *z, double *val, double kappa)

Not implemented yet

Global `Vgreen_helmholtzD`(Vgreen *thee, int npos, double *x, double *y, double *z, double *gradx, double *grady, dou

Not implemented yet

Global `Vgrid_writeUHBD`(Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char

This routine does not respect partition information

Global `Vpbe_ctor2`(Vpbe *thee, Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, d

The focusing flag is currently not used!!

Global `Vpee_markRefine`(Vpee *thee, AM *am, int level, int akey, int rcol, double etol, int bkey)

This function is no longer up-to-date with FEtk and may not function properly

Global `Vpmg_printColComp`(Vpmg *thee, char path[72], char title[72], char mxtyp[3], int flag)

Can this path variable be replaced with a Vio socket?

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Vcsm class	??
Vfetk class	??
Vpee class	??
APOLparm class	??
FEMparm class	??
MGparm class	??
NOsh class	??
PBEParm class	??
Vacc class	??
Valist class	??
Vatom class	??
Vcap class	??
Vclist class	??
Vgreen class	??
Vhal class	??
Vparam class	??
Vpbe class	??
Vstring class	??
Vunit class	??
Vgrid class	??
Vmgrid class	??
Vopot class	??
Vpmg class	??
Vpmgp class	??

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

<code>sAPOLparm</code> (Parameter structure for APOL-specific variables from input files)	??
<code>sFEMparm</code> (Parameter structure for FEM-specific variables from input files)	??
<code>sMGparm</code> (Parameter structure for MG-specific variables from input files)	??
<code>sNOsh</code> (Class for parsing fixed format input files)	??
<code>sNOsh_calc</code> (Calculation class for use when parsing fixed format input files)	??
<code>sPBEparm</code> (Parameter structure for PBE variables from input files)	??
<code>sVacc</code> (Oracle for solvent- and ion-accessibility around a biomolecule)	??
<code>sVaccSurf</code> (Surface object list of per-atom surface points)	??
<code>sValist</code> (Container class for list of atom objects)	??
<code>sVatom</code> (Contains public data members for Vatom class/module)	??
<code>sVclist</code> (Atom cell list)	??
<code>sVclistCell</code> (Atom cell list cell)	??
<code>sVcsm</code> (Charge-simplex map class)	??
<code>sVfetk</code> (Contains public data members for Vfetk class/module)	??
<code>sVfetk_LocalVar</code> (Vfetk LocalVar subclass)	??
<code>sVgreen</code> (Contains public data members for Vgreen class/module)	??
<code>sVgrid</code> (Electrostatic potential oracle for Cartesian mesh data)	??
<code>sVmgrid</code> (Multiresoltion oracle for Cartesian mesh data)	??
<code>sVopot</code> (Electrostatic potential oracle for Cartesian mesh data)	??
<code>sVparam_AtomData</code> (AtomData sub-class; stores atom data)	??
<code>sVpbe</code> (Contains public data members for Vpbe class/module)	??
<code>sVpee</code> (Contains public data members for Vpee class/module)	??
<code>sVpmg</code> (Contains public data members for Vpmg class/module)	??
<code>sVpmgp</code> (Contains public data members for Vpmgp class/module)	??
<code>Vparam</code> (Reads and assigns charge/radii parameters)	??

[Vparam_ResData](#) (ResData sub-class; stores residue data) ??

Chapter 7

File Index

7.1 File List

Here is a list of all documented files with brief descriptions:

```
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/doc/license/LICENSE.h (APBS license ) . . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/doc/programmer/mainpage.h . . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/aaa_-
    inc/apbs/apbs.h (Top-level header for APBS ) . . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/aaa_-
    lib/apbs\_link.c (Autoconf linkage assistance for packages built on top
        of APBS ) . . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/src/fem/dummy.c (Give libtool something to do ) . . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/src/fem/vcsm.c (Class Vcsm methods ) . . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/src/fem/vfetk.c (Class Vfetk methods ) . . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/src/fem/vpee.c (Class Vpee methods ) . . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/src/fem/apbs/vcsm.h (Contains declarations for the Vcsm class ) . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/src/fem/apbs/vfetk.h (Contains declarations for class Vfetk ) . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/src/fem/apbs/vpee.h (Contains declarations for class Vpee ) . . . . ???
C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
    S/src/generic/apolparm.c (Class APOLparm methods ) . . . . . ???
```

C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[femparm.c](#) (Class FEMparm methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[mgparm.c](#) (Class MGparm methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[nosh.c](#) (Class NOsh methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[pbeparm.c](#) (Class PBEparm methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[vacc.c](#) (Class Vacc methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[valist.c](#) (Class Valist methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[vatom.c](#) (Class Vatom methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[vcap.c](#) (Class Vcap methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[vclist.c](#) (Class Vclist methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[vgreen.c](#) (Class Vgreen methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[vparam.c](#) (Class Vparam methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[vpbe.c](#) (Class Vpbe methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/[vstring.c](#) (Class Vstring methods) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/apbs/[apolparm.h](#) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/apbs/[femparm.h](#) (Contains declarations for class APOL-
 parm) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/apbs/[mgparm.h](#) (Contains declarations for class MG-
 parm) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/apbs/[nosh.h](#) (Contains declarations for class NOsh) . . ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/apbs/[pbeparm.h](#) (Contains declarations for class PBEparm
) ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/apbs/[vacc.h](#) (Contains declarations for class Vacc) . . ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/apbs/[valist.h](#) (Contains declarations for class Valist) . . ??
 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
 S/src/generic/apbs/[vatom.h](#) (Contains declarations for class Vatom) ??

Chapter 8

Module Documentation

8.1 Vcsm class

A charge-simplex map for evaluating integrals of delta functions in a finite element setting.

Data Structures

- struct [sVcsm](#)

Charge-simplex map class.

Files

- file [vcsm.h](#)

Contains declarations for the Vcsm class.

- file [vcsm.c](#)

Class Vcsm methods.

Typedefs

- typedef struct [sVcsm](#) [Vcsm](#)

Declaration of the Vcsm class as the Vcsm structure.

Functions

- VEXTERNC void [Gem_setExternalUpdateFunction](#) (Gem *thee, void(*externalUpdate)(SS **simps, int num))

External function for FEtk Gem class to use during mesh refinement.
- VEXTERNC Valist * [Vcsm_getValist](#) (Vcsm *thee)

Get atom list.
- VEXTERNC int [Vcsm_getNumberAtoms](#) (Vcsm *thee, int isimp)

Get number of atoms associated with a simplex.
- VEXTERNC Vatom * [Vcsm_getAtom](#) (Vcsm *thee, int iatom, int isimp)

Get particular atom associated with a simplex.
- VEXTERNC int [Vcsm_getAtomIndex](#) (Vcsm *thee, int iatom, int isimp)

Get ID of particular atom in a simplex.
- VEXTERNC int [Vcsm_getNumberSimplices](#) (Vcsm *thee, int iatom)

Get number of simplices associated with an atom.
- VEXTERNC SS * [Vcsm_getSimplex](#) (Vcsm *thee, int isimp, int iatom)

Get particular simplex associated with an atom.
- VEXTERNC int [Vcsm_getSimplexIndex](#) (Vcsm *thee, int isimp, int iatom)

Get index particular simplex associated with an atom.
- VEXTERNC unsigned long int [Vcsm_memChk](#) (Vcsm *thee)

Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC Vcsm * [Vcsm_ctor](#) (Valist *alist, Gem *gm)

Construct Vcsm object.
- VEXTERNC int [Vcsm_ctor2](#) (Vcsm *thee, Valist *alist, Gem *gm)

FORTRAN stub to construct Vcsm object.
- VEXTERNC void [Vcsm_dtor](#) (Vcsm **thee)

Destroy Vcsm object.
- VEXTERNC void [Vcsm_dtor2](#) (Vcsm *thee)

FORTRAN stub to destroy Vcsm object.
- VEXTERNC void [Vcsm_init](#) (Vcsm *thee)

Initialize charge-simplex map with mesh and atom data.
- VEXTERNC int [Vcsm_update](#) (Vcsm *thee, SS **simps, int num)

Update the charge-simplex and simplex-charge maps after refinement.

8.1.1 Detailed Description

A charge-simplex map for evaluating integrals of delta functions in a finite element setting.

8.1.2 Function Documentation

8.1.2.1 VEXTERNC void Gem_setExternalUpdateFunction (*Gem * thee, void(*)(SS **simps, int num) externalUpdate*)

External function for FEtk Gem class to use during mesh refinement.

Author

Nathan Baker

Parameters

<i>thee</i>	The FEtk geometry manager
<i>externalUpdate</i>	Function pointer for call during mesh refinement

Here is the caller graph for this function:



8.1.2.2 VEXTERNC Vcsm* Vcsm_ctor (*Valist * alist, Gem * gm*)

Construct Vcsm object.

Author

Nathan Baker

Note

- The initial mesh must be sufficiently coarse for the assignment procedures to be efficient
- The map is not built until Vcsm_init is called

Returns

Pointer to newly allocated Vcsm object

Parameters

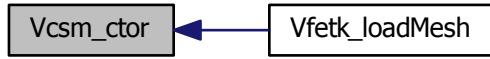
<i>alist</i>	List of atoms
<i>gm</i>	FEtk geometry manager defining the mesh

Definition at line 140 of file [vcsrm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.2.3 VEXTERNC int Vcsm_ctor2 (Vcsm * *thee*, Valist * *alist*, Gem * *gm*)

FORTRAN stub to construct Vcsm object.

Author

Nathan Baker

Note

- The initial mesh must be sufficiently coarse for the assignment procedures to be efficient
- The map is not built until Vcsm_init is called

Returns

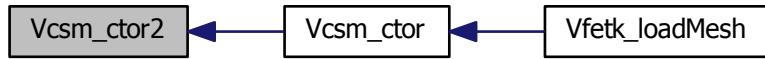
1 if successful, 0 otherwise

Parameters

<i>thee</i>	The Vcsm object
<i>alist</i>	The list of atoms
<i>gm</i>	The FETk geometry manager defining the mesh

Definition at line 151 of file [vcsm.c](#).

Here is the caller graph for this function:

**8.1.2.4 VEXTERNC void Vcsm_dtor (Vcsm ** *thee*)**

Destroy Vcsm object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location for Vcsm object
-------------	--

Definition at line 292 of file [vcsm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.2.5 VEXTERNC void Vcsm_dtor2 (**Vcsm** * *thee*)

FORTRAN stub to destroy Vcsm object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vcsm object
-------------	------------------------

Definition at line 300 of file [vcsms.c](#).

Here is the caller graph for this function:



8.1.2.6 VEXTERNC Vatom* Vcsm_getAtom (Vcsm * *thee*, int *iatom*, int *isimp*)

Get particular atom associated with a simplex.

Author

Nathan Baker

Returns

Array of atoms associated with a simplex

Parameters

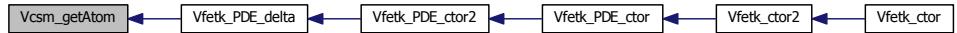
<i>thee</i>	The Vcsm object
<i>iatom</i>	Index of atom in Vcsm list ofr this simplex
<i>isimp</i>	Simplex ID

Definition at line 81 of file [vcsm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.2.7 VEXTERNC int Vcsm_getAtomIndex (*Vcsm * thee, int iatom, int isimp*)

Get ID of particular atom in a simplex.

Author

Nathan Baker

Returns

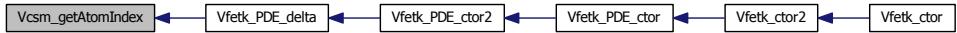
Index of atom in Valist object

Parameters

<i>thee</i>	The Vcsm object
<i>iatom</i>	Index of atom in Vcsm list for this simplex
<i>isimp</i>	Simplex ID

Definition at line 92 of file [vcsms.c](#).

Here is the caller graph for this function:



8.1.2.8 VEXTERNC int Vcsm_getNumberAtoms (*Vcsm * thee, int isimp*)

Get number of atoms associated with a simplex.

Author

Nathan Baker

Returns

Number of atoms associated with a simplex

Parameters

<i>thee</i>	The Vcsm object
<i>isimp</i>	Simplex ID

Definition at line 73 of file [vcsm.c](#).

Here is the caller graph for this function:



8.1.2.9 VEXTERNC int Vcsm_getNumberSimplices (*Vcsm* * *thee*, int *iatom*)

Get number of simplices associated with an atom.

Author

Nathan Baker

Returns

Number of simplices associated with an atom

Parameters

<i>thee</i>	The Vcsm object
<i>iatom</i>	The Valist atom index

Definition at line 103 of file [vcsm.c](#).

8.1.2.10 VEXTERNC SS* Vcsm_getSimplex (*Vcsm* * *thee*, int *isimp*, int *iatom*)

Get particular simplex associated with an atom.

Author

Nathan Baker

Returns

Pointer to simplex object

Parameters

<i>thee</i>	The Vcsm object
<i>isimp</i>	Index of simplex in Vcsm list
<i>iatom</i>	Valist atom index

Definition at line 113 of file [vcsms.c](#).

Here is the caller graph for this function:



8.1.2.11 VEXTERNC int Vcsm_getSimplexIndex (Vcsm * *thee*, int *isimp*, int *iatom*)

Get index particular simplex associated with an atom.

Author

Nathan Baker

Returns

Gem index of specified simplex

Parameters

<i>thee</i>	The Vcsm object
<i>isimp</i>	Index of simplex in Vcsm list
<i>iatom</i>	Index of atom in Valist

Definition at line 123 of file [vcsms.c](#).

8.1.2.12 VEXTERNC Valist* Vcsm_getValist (Vcsm * *thee*)

Get atom list.

Author

Nathan Baker

Returns

Pointer to Valist atom list

Parameters

<i>thee</i>	The Vcsm object
-------------	-----------------

Definition at line 66 of file [vcsms.c](#).

8.1.2.13 VEXTERNC void Vcsm_init (Vcsm * *thee*)

Initialize charge-simplex map with mesh and atom data.

Author

Nathan Baker

Note

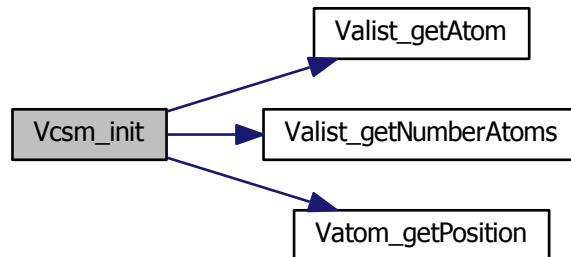
The initial mesh must be sufficiently coarse for the assignment procedures to be efficient

Parameters

<i>thee</i>	The Vcsm object
-------------	-----------------

Definition at line 174 of file [vcsms.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.2.14 VEXTERNC unsigned long int `Vcsm_memChk` (`Vcsm * thee`)

Return the memory used by this structure (and its contents) in bytes.

Author

Nathan Baker

Returns

The memory used by this structure and its contents in bytes

Parameters

<code>thee</code>	The <code>Vcsm</code> object
-------------------	------------------------------

Definition at line 133 of file [vcsms.c](#).

Here is the caller graph for this function:



8.1.2.15 VEXTERNC int Vcsm_update (*Vcsm *thee*, *SS **simps*, *int num*)

Update the charge-simplex and simplex-charge maps after refinement.

Author

Nathan Baker

Returns

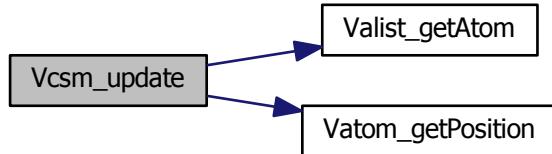
1 if successful, 0 otherwise

Parameters

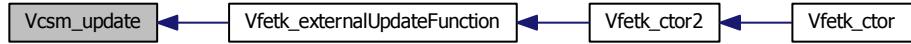
<i>thee</i>	The Vcsm object
<i>simps</i>	List of pointer to newly created (by refinement) simplex objects. The first simplex is expected to be derived from the parent simplex and therefore have the same ID. The remaining simplices are the children and should represent new entries in the charge-simplex map.
<i>num</i>	Number of simplices in simps list

Definition at line 326 of file [vcsms.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2 Vfetk class

FEtk master class (interface between FEtk and APBS)

Data Structures

- struct [sVfetk](#)
Contains public data members for Vfetk class/module.
- struct [sVfetk_LocalVar](#)
Vfetk LocalVar subclass.

Files

- file [vfetk.h](#)
Contains declarations for class Vfetk.

- file [vfetk.c](#)

Class Vfetk methods.

Defines

- `#define VRINGMAX 1000`
Maximum number of simplices in a simplex ring.
- `#define VATOMMAX 1000000`
Maximum number of atoms associated with a vertex.

TypeDefs

- `typedef enum eVfetk_LsolvType Vfetk_LsolvType`
Declare FEMparm_LsolvType type.
- `typedef enum eVfetk_MeshLoad Vfetk_MeshLoad`
Declare FEMparm_GuessType type.
- `typedef enum eVfetk_NsolvType Vfetk_NsolvType`
Declare FEMparm_NsolvType type.
- `typedef enum eVfetk_GuessType Vfetk_GuessType`
Declare FEMparm_GuessType type.
- `typedef enum eVfetk_PrecType Vfetk_PrecType`
Declare FEMparm_GuessType type.
- `typedef struct sVfetk_LocalVar Vfetk_LocalVar`
Declaration of the Vfetk_LocalVar subclass as the Vfetk_LocalVar structure.
- `typedef struct sVfetk Vfetk`
Declaration of the Vfetk class as the Vfetk structure.

Enumerations

- `enum eVfetk_LsolvType { VLT_SLU = 0, VLT_MG = 1, VLT(CG) = 2, VLT_BCG = 3 }`
Linear solver type.
- `enum eVfetk_MeshLoad { VML_DIRICUBE, VML_NEUMCUBE, VML_EXTERNAL }`
Mesh loading operation.
- `enum eVfetk_NsolvType { VNT_NEW = 0, VNT_INC = 1, VNT_ARC = 2 }`
Non-linear solver type.
- `enum eVfetk_GuessType { VGT_ZERO = 0, VGT_DIRI = 1, VGT_PREV = 2 }`
Initial guess type.
- `enum eVfetk_PrecType { VPT_IDEN = 0, VPT_DIAG = 1, VPT_MG = 2 }`
Preconditioner type.

Functions

- VEXTERNC `Gem * Vfetk_getGem (Vfetk *thee)`
Get a pointer to the Gem (grid manager) object.
- VEXTERNC `AM * Vfetk_getAM (Vfetk *thee)`
Get a pointer to the AM (algebra manager) object.
- VEXTERNC `Vpbe * Vfetk_getVpbe (Vfetk *thee)`
Get a pointer to the Vpbe (PBE manager) object.
- VEXTERNC `Vcsm * Vfetk_getVcsm (Vfetk *thee)`
Get a pointer to the Vcsm (charge-simplex map) object.
- VEXTERNC `int Vfetk_getAtomColor (Vfetk *thee, int iatom)`
Get the partition information for a particular atom.
- VEXTERNC `Vfetk * Vfetk_ctor (Vpbe *pbe, Vhal_PBEType type)`
Constructor for Vfetk object.
- VEXTERNC `int Vfetk_ctor2 (Vfetk *thee, Vpbe *pbe, Vhal_PBEType type)`
FORTRAN stub constructor for Vfetk object.
- VEXTERNC `void Vfetk_dtor (Vfetk **thee)`
Object destructor.
- VEXTERNC `void Vfetk_dtor2 (Vfetk *thee)`
FORTRAN stub object destructor.
- VEXTERNC `double * Vfetk_getSolution (Vfetk *thee, int *length)`
Create an array containing the solution (electrostatic potential in units of $k_B T / e$) at the finest mesh level.
- VEXTERNC `void Vfetk_setParameters (Vfetk *thee, PBEparm *pbeparm, FEM-parm *feparm)`
Set the parameter objects.
- VEXTERNC `double Vfetk_energy (Vfetk *thee, int color, int nonlin)`
Return the total electrostatic energy.
- VEXTERNC `double Vfetk_dqmEnergy (Vfetk *thee, int color)`
Get the "mobile charge" and "polarization" contributions to the electrostatic energy.
- VEXTERNC `double Vfetk_qfEnergy (Vfetk *thee, int color)`
Get the "fixed charge" contribution to the electrostatic energy.
- VEXTERNC `unsigned long int Vfetk_memChk (Vfetk *thee)`
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC `void Vfetk_setAtomColors (Vfetk *thee)`
Transfer color (partition ID) information from a partitioned mesh to the atoms.
- VEXTERNC `Bmat_printHB (Bmat *thee, char *fname)`
Writes a Bmat to disk in Harwell-Boeing sparse matrix format.
- VEXTERNC `Vrc_Codes Vfetk_genCube (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType)`

Construct a rectangular mesh (in the current Vfetk object)

- VEXTERNC `Vrc_Codes` `Vfetk_loadMesh` (`Vfetk` *`thee`, double `center[3]`, double `length[3]`, `Vfetk_MeshLoad` `meshType`, `Vio` *`sock`)
Loads a mesh into the Vfetk (and associated) object(s).
- VEXTERNC `PDE` * `Vfetk_PDE_ctor` (`Vfetk` *`fetk`)
Constructs the FEtk PDE object.
- VEXTERNC `int` `Vfetk_PDE_ctor2` (`PDE` *`thee`, `Vfetk` *`fetk`)
Initializes the FEtk PDE object.
- VEXTERNC `void` `Vfetk_PDE_dtor` (`PDE` **`thee`)
Destroys FEtk PDE object.
- VEXTERNC `void` `Vfetk_PDE_dtor2` (`PDE` *`thee`)
FORTRAN stub: destroys FEtk PDE object.
- VEXTERNC `void` `Vfetk_PDE_initAssemble` (`PDE` *`thee`, `int` `ip[]`, double `rp[]`)
Do once-per-assembly initialization.
- VEXTERNC `void` `Vfetk_PDE_initElement` (`PDE` *`thee`, `int` `elementType`, `int` `chart`, double `txv[]`[`VAPBS_DIM`], `void` *`data`)
Do once-per-element initialization.
- VEXTERNC `void` `Vfetk_PDE_initFace` (`PDE` *`thee`, `int` `faceType`, `int` `chart`, double `tnvec[]`)
Do once-per-face initialization.
- VEXTERNC `void` `Vfetk_PDE_initPoint` (`PDE` *`thee`, `int` `pointType`, `int` `chart`, double `txq[]`, double `tU[]`, double `tdU[]`[`VAPBS_DIM`])
Do once-per-point initialization.
- VEXTERNC `void` `Vfetk_PDE_Fu` (`PDE` *`thee`, `int` `key`, double `F[]`)
Evaluate strong form of PBE. For interior points, this is:

$$-\nabla \cdot \epsilon \nabla u + b(u) - f$$

where $b(u)$ is the (possibly nonlinear) mobile ion term and f is the source charge distribution term (for PBE) or the induced surface charge distribution (for RPBE). For an interior-boundary (simplex face) point, this is:

$$[\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^+} - [\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^-}$$

where $n(x)$ is the normal to the simplex face and the term represents the jump in dielectric displacement across the face. There is no outer-boundary contribution for this problem.

- VEXTERNC `double` `Vfetk_PDE_Fu_v` (`PDE` *`thee`, `int` `key`, double `V[]`, double `dV[]`[`VAPBS_DIM`])

This is the weak form of the PBE; i.e. the strong form integrated with a test function to give:

$$\int_{\Omega} [\epsilon \nabla u \cdot \nabla v + b(u)v - fv] dx$$

where $b(u)$ denotes the mobile ion term.

- VEXTERNC double [Vfetk_PDE_DFu_wv](#) (PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[][VAPBS_DIM])

*This is the linearization of the weak form of the PBE; e.g., for use in a Newton iteration.
This is the functional linearization of the strong form integrated with a test function to give:*

$$\int_{\Omega} [\epsilon \nabla w \cdot \nabla v + b'(u) w v - f v] dx$$

where $b'(u)$ denotes the functional derivation of the mobile ion term.

- VEXTERNC void [Vfetk_PDE_delta](#) (PDE *thee, int type, int chart, double txq[], void *user, double F[])

Evaluate a (discretized) delta function source term at the given point.

- VEXTERNC void [Vfetk_PDE_u_D](#) (PDE *thee, int type, int chart, double txq[], double F[])

Evaluate the Dirichlet boundary condition at the given point.

- VEXTERNC void [Vfetk_PDE_u_T](#) (PDE *thee, int type, int chart, double txq[], double F[])

Evaluate the "true solution" at the given point for comparison with the numerical solution.

- VEXTERNC void [Vfetk_PDE_bisectEdge](#) (int dim, int dimll, int edgeType, int chart[], double vx[][VAPBS_DIM])

Define the way manifold edges are bisected.

- VEXTERNC void [Vfetk_PDE_mapBoundary](#) (int dim, int dimll, int vertexType, int chart, double vx[VAPBS_DIM])

Map a boundary point to some pre-defined shape.

- VEXTERNC int [Vfetk_PDE_markSimplex](#) (int dim, int dimll, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][VAPBS_DIM], void *simplex)

User-defined error estimator -- in our case, a geometry-based refinement method; forcing simplex refinement at the dielectric boundary and (for non-regularized PBE) the charges.

- VEXTERNC void [Vfetk_PDE_oneChart](#) (int dim, int dimll, int objType, int chart[], double vx[][VAPBS_DIM], int dimV)

Unify the chart for different coordinate systems -- a no-op for us.

- VEXTERNC double [Vfetk_PDE_Ju](#) (PDE *thee, int key)

Energy functional. This returns the energy (less delta function terms) in the form:

$$c^{-1}/2 \int (\epsilon(\nabla u)^2 + \kappa^2(cosh u - 1)) dx$$

for a 1:1 electrolyte where c is the output from `Vpbe_getZmagic`.

- VEXTERNC void [Vfetk_externalUpdateFunction](#) (SS **simps, int num)

External hook to simplex subdivision routines in Gem. Called each time a simplex is subdivided (we use it to update the charge-simplex map)

- VEXTERNC int [Vfetk_PDE_simplexBasisInit](#) (int key, int dim, int comp, int *ndof, int dof[])

Initialize the bases for the trial or the test space, for a particular component of the system, at all quadrature points on the master simplex element.

- VEXTERNC void [Vfetk_PDE_simplexBasisForm](#) (int key, int dim, int comp, int pdkey, double xq[], double basis[])

Evaluate the bases for the trial or test space, for a particular component of the system, at all quadrature points on the master simplex element.

- VEXTERNC void [Vfetk_readMesh](#) (Vfetk *thee, int skey, Vio *sock)

Read in mesh and initialize associated internal structures.

- VEXTERNC void [Vfetk_dumpLocalVar](#) ()

Debugging routine to print out local variables used by PDE object.

- VEXTERNC int [Vfetk_fillArray](#) (Vfetk *thee, Bvec *vec, [Vdata_Type](#) type)

Fill an array with the specified data.

- VEXTERNC int [Vfetk_write](#) (Vfetk *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, Bvec *vec, [Vdata_Format](#) format)

Write out data.

- VEXTERNC Vrc_Codes [Vfetk_loadGem](#) (Vfetk *thee, Gem *gm)

Load a Gem geometry manager object into Vfetk.

8.2.1 Detailed Description

FEtk master class (interface between FEtk and APBS)

8.2.2 Enumeration Type Documentation

8.2.2.1 enum eVfetk_GuessType

Initial guess type.

Note

Do not change these values; they correspond to settings in FEtk

Enumerator:

VGT_ZERO Zero initial guess

VGT_DIRI Dirichlet boundary condition initial guess

VGT_PREV Previous level initial guess

Definition at line 135 of file [vfetk.h](#).

8.2.2.2 enum eVfetk_LsolvType

Linear solver type.

Note

Do not change these values; they correspond to settings in FETk

Enumerator:

- VLT_SLU** SuperLU direct solve
- VLT_MG** Multigrid
- VLT(CG)** Conjugate gradient
- VLT_BCG** BiCGStab

Definition at line 83 of file [vfetk.h](#).

8.2.2.3 enum eVfetk_MeshLoad

Mesh loading operation.

Enumerator:

- VML_DIRICUBE** Dirichlet cube
- VML_NEUMCUBE** Neumann cube
- VML_EXTERNAL** External mesh (from socket)

Definition at line 101 of file [vfetk.h](#).

8.2.2.4 enum eVfetk_NsolvType

Non-linear solver type.

Note

Do not change these values; they correspond to settings in FETk

Enumerator:

- VNT_NEW** Newton solver
- VNT_INC** Incremental
- VNT_ARC** Psuedo-arclength

Definition at line 118 of file [vfetk.h](#).

8.2.2.5 enum eVfetk_PrecType

Preconditioner type.

Note

Do not change these values; they correspond to settings in FEtk

Enumerator:

VPT_IDEN Identity matrix

VPT_DIAG Diagonal scaling

VPT_MG Multigrid

Definition at line [152](#) of file [vfetk.h](#).

8.2.3 Function Documentation

8.2.3.1 VEXTERNC void Bmat_printHB (Bmat * *thee*, char * *fname*)

Writes a Bmat to disk in Harwell-Boeing sparse matrix format.

Author

Stephen Bond

Note

This is a friend function of Bmat

Bug

Hardwired to only handle the single block symmetric case.

Parameters

<i>thee</i>	The matrix to write
<i>fname</i>	Filename for output

Definition at line [966](#) of file [vfetk.c](#).

8.2.3.2 VEXTERNC Vfetk* Vfetk_ctor (Vpbe * *pbe*, Vhal_PBEType *type*)

Constructor for Vfetk object.

Author

Nathan Baker

Returns

Pointer to newly allocated Vfetk object

Note

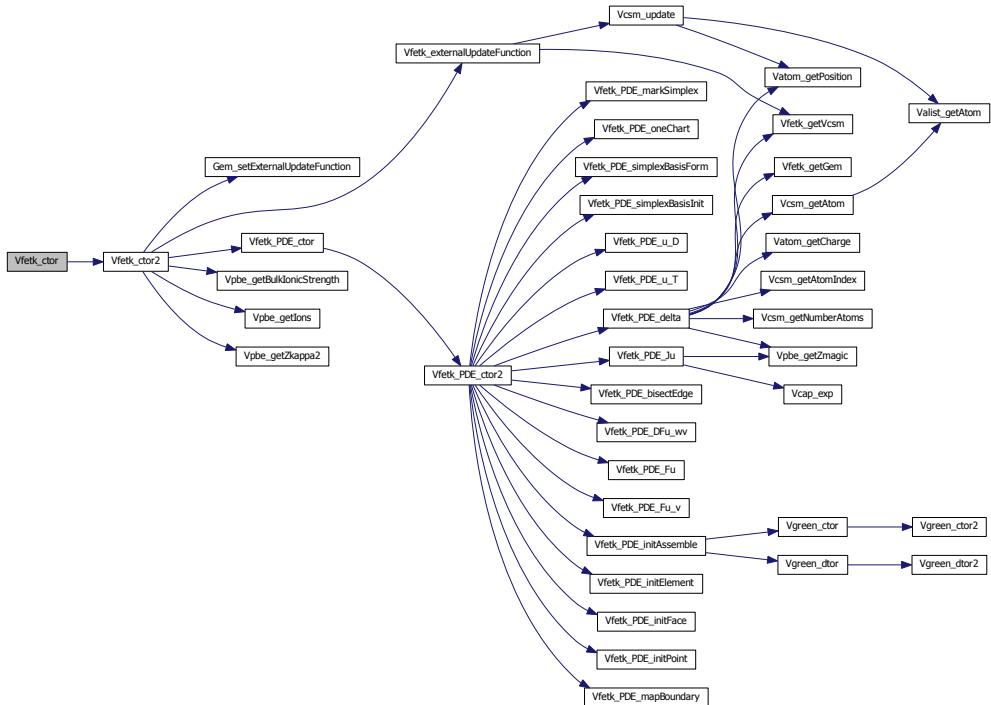
This sets up the Gem, AM, and Aprx FEtk objects but does not create a mesh. The easiest way to create a mesh is to then call `Vfetk_genCube`

Parameters

<i>pbe</i>	Vpbe (PBE manager object)
<i>type</i>	Version of PBE to solve

Definition at line 535 of file [vfetk.c](#).

Here is the call graph for this function:



8.2.3.3 VEXTERNC int Vfetk_ctor2 (*Vfetk * thee*, *Vpbe * pbe*, *Vhal_PBEType type*)

FORTRAN stub constructor for Vfetk object.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Note

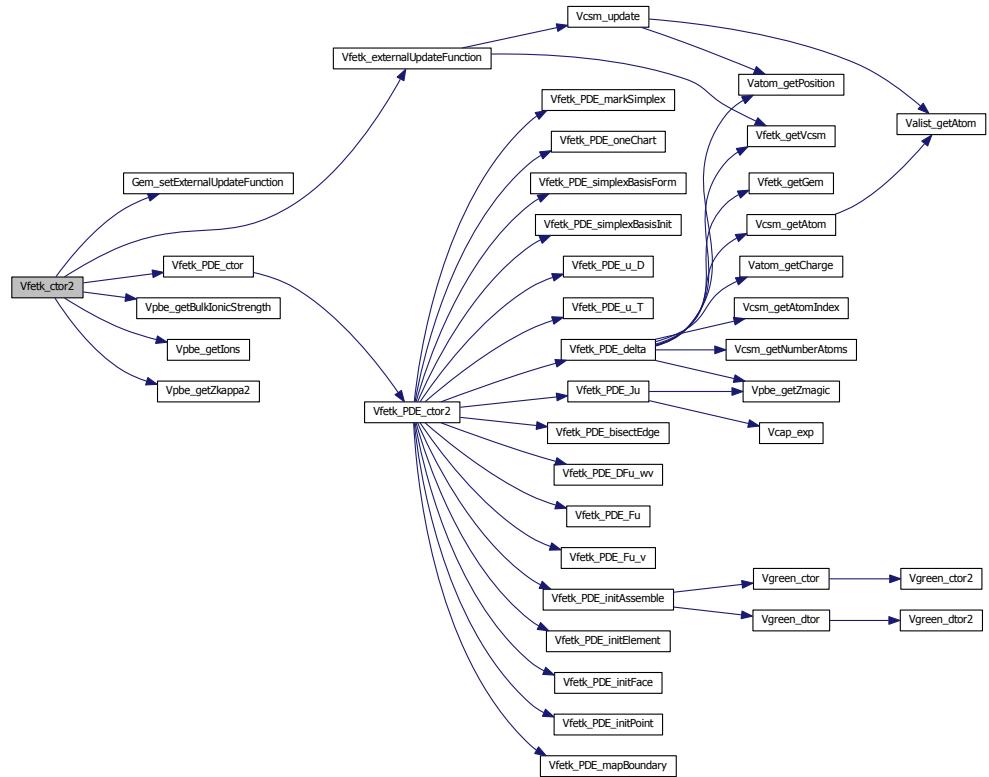
This sets up the Gem, AM, and Aprx FETk objects but does not create a mesh. The easiest way to create a mesh is to then call Vfetk_genCube

Parameters

<i>thee</i>	Vfetk object memory
<i>pbe</i>	PBE manager object
<i>type</i>	Version of PBE to solve

Definition at line 546 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3.4 VEXTERNC double Vfetk_dqmEnergy (*Vfetk * thee, int color*)

Get the "mobile charge" and "polarization" contributions to the electrostatic energy.

Using the solution at the finest mesh level, get the electrostatic energy due to the interaction of the mobile charges with the potential and polarization of the dielectric medium:

$$G = \frac{1}{4I_s} \sum_i c_i q_i^2 \int \bar{\kappa}^2(x) e^{-q_i u(x)} dx + \frac{1}{2} \int \epsilon (\nabla u)^2 dx$$

for the NPBE and

$$G = \frac{1}{2} \int \bar{\kappa}^2(x) u^2(x) dx + \frac{1}{2} \int \epsilon (\nabla u)^2 dx$$

for the LPBE. Here i denotes the counterion species, I_s is the bulk ionic strength, $\bar{\kappa}^2(x)$ is the modified Debye-Huckel parameter, c_i is the concentration of species i , q_i is the charge of species i , ϵ is the dielectric function, and $u(x)$ is the dimensionless electrostatic potential. The energy is scaled to units of $k_b T$.

Author

Nathan Baker

Parameters

<i>thee</i>	Vfetk object
<i>color</i>	Partition restriction for energy evaluation, only used if non-negative

Returns

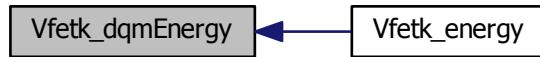
The "mobile charge" and "polarization" contributions to the electrostatic energy in units of $k_b T$.

Parameters

<i>thee</i>	The Vfetk object
<i>color</i>	Partition restriction for energy calculation; if non-negative, energy calculation is restricted to the specified partition (indexed by simplex and atom colors)

Definition at line 802 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.5 VEXTERNC void Vfetk_dtor (*Vfetk* ** *thee*)

Object destructor.

Author

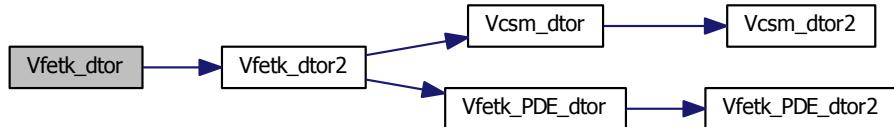
Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of Vfetk object
-------------	--

Definition at line 621 of file [vfetk.c](#).

Here is the call graph for this function:



8.2.3.6 VEXTERNC void Vfetk_dtor2 (*Vfetk* * *thee*)

FORTRAN stub object destructor.

Author

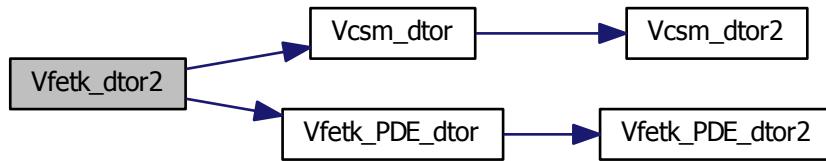
Nathan Baker

Parameters

<i>thee</i>	Pointer to Vfetk object to be destroyed
-------------	---

Definition at line [629](#) of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3.7 VEXTERNC void Vfetk_dumpLocalVar()

Debugging routine to print out local variables used by PDE object.

Author

Nathan Baker

Bug

This function is not thread-safe

Definition at line [2178](#) of file [vfetk.c](#).

8.2.3.8 VEXTERNC double Vfetk.energy (*Vfetk * thee, int color, int nonlin*)

Return the total electrostatic energy.

Using the solution at the finest mesh level, get the electrostatic energy using the free energy functional for the Poisson-Boltzmann equation without removing any self-interaction terms (i.e., removing the reference state of isolated charges present in an infinite dielectric continuum with the same relative permittivity as the interior of the protein) and return the result in units of $k_B T$. The argument color allows the user to control the partition on which this energy is calculated; if (color == -1) no restrictions are used. The solution is obtained from the finest level of the passed AM object, but atomic data from the Vfetk object is used to calculate the energy.

Author

Nathan Baker

Returns

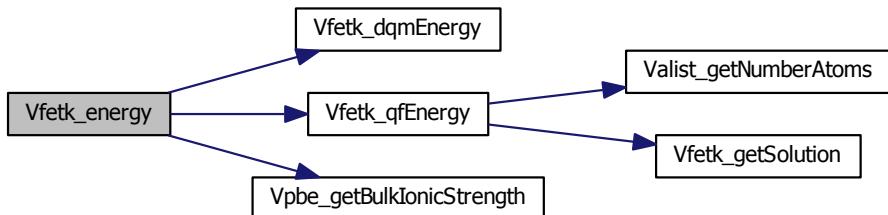
Total electrostatic energy in units of $k_B T$.

Parameters

<i>thee</i>	THe Vfetk object
<i>color</i>	Partition restriction for energy calculation; if non-negative, energy calculation is restricted to the specified partition (indexed by simplex and atom colors)
<i>nonlin</i>	If 1, the NPBE energy functional is used; otherwise, the LPBE energy functional is used. If -2, SMPBE is used.

Definition at line [668](#) of file [vfetk.c](#).

Here is the call graph for this function:



8.2.3.9 VEXTERNC void Vfetk_externalUpdateFunction (SS ** *simps*, int *num*)

External hook to simplex subdivision routines in Gem. Called each time a simplex is subdivided (we use it to update the charge-simplex map)

Author

Nathan Baker

Bug

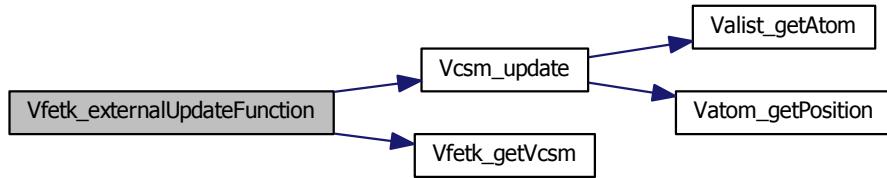
This function is not thread-safe.

Parameters

<i>simps</i>	List of parent (<code>simps[0]</code>) and children (remainder) simplices
<i>num</i>	Number of simplices in list

Definition at line 2001 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3.10 VEXTERNC int Vfetk_fillArray (`Vfetk * thee`, `Bvec * vec`, `Vdata_Type type`)

Fill an array with the specified data.

Author

Nathan Baker

Note

This function is thread-safe

Bug

Several values of type are not implemented

Returns

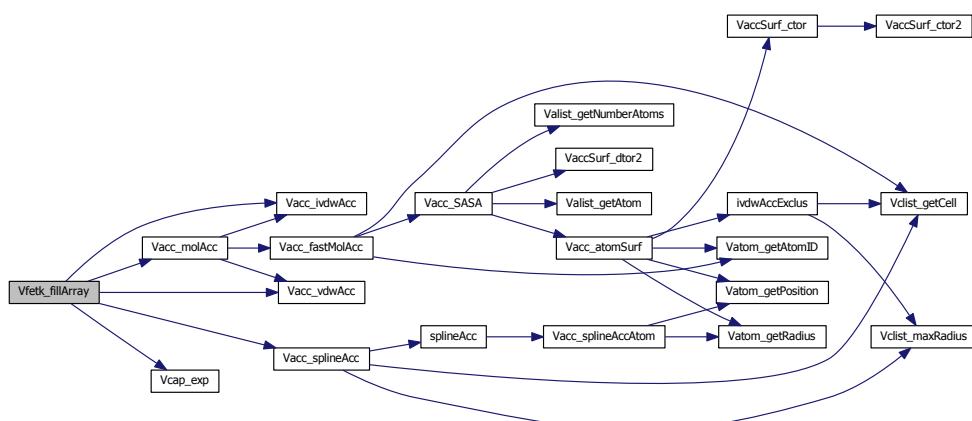
1 if successful, 0 otherwise

Parameters

<i>thee</i>	The Vfetk object with the data
<i>vec</i>	The vector to hold the data
<i>type</i>	THe type of data to write

Definition at line 2222 of file vfetk.c.

Here is the call graph for this function:



8.2.3.11 VEXTERNC Vrc_Codes Vfetk_genCube (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType)

Construct a rectangular mesh (in the current Vfetk object)

Author

Nathan Baker

Parameters

<i>thee</i>	Vfetk object
<i>center</i>	Center for mesh
<i>length</i>	Mesh lengths
<i>meshType</i>	Mesh boundary conditions

Definition at line 838 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.12 VEXTERNC AM* Vfetk_getAM (Vfetk * *thee*)

Get a pointer to the AM (algebra manager) object.

Author

Nathan Baker

Returns

Pointer to the AM (algebra manager) object

Parameters

<i>thee</i>	The Vfetk object
-------------	------------------

Definition at line 502 of file [vfetk.c](#).

8.2.3.13 VEXTERNC int Vfetk_getAtomColor (Vfetk * *thee*, int *iatom*)

Get the partition information for a particular atom.

Author

Nathan Baker

Note

Friend function of Vatom

Returns

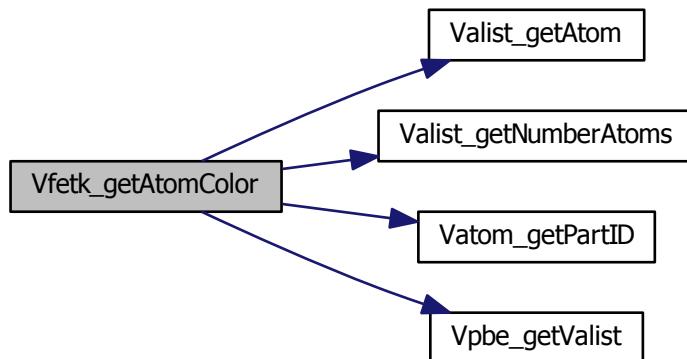
Partition ID

Parameters

<i>thee</i>	The Vfetk object
<i>iatom</i>	Valist atom index

Definition at line 522 of file [vfetk.c](#).

Here is the call graph for this function:



8.2.3.14 VEXTERNC Gem* Vfetk_getGem (Vfetk * *thee*)

Get a pointer to the Gem (grid manager) object.

Author

Nathan Baker

Returns

Pointer to the Gem (grid manager) object

Parameters

<i>thee</i>	Vfetk object
-------------	--------------

Definition at line 495 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.15 VEXTERNC double* Vfetk_getSolution (Vfetk * *thee*, int * *length*)

Create an array containing the solution (electrostatic potential in units of $k_B T / e$) at the finest mesh level.

Author

Nathan Baker and Michael Holst

Note

The user is responsible for destroying the newly created array

Returns

Newly created array of length "length" (see above); the user is responsible for destruction

Parameters

<i>thee</i>	Vfetk object with solution
<i>length</i>	Ste to length of the newly created solution array

Definition at line 637 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.16 VEXTERNC Vcsm* Vfetk_getVcsm (Vfetk * *thee*)

Get a pointer to the Vcsm (charge-simplex map) object.

Author

Nathan Baker

Returns

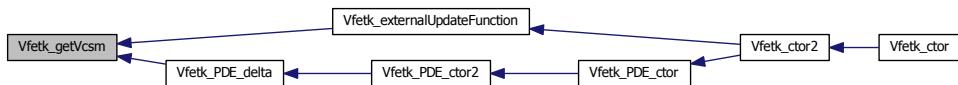
Pointer to the Vcsm (charge-simplex map) object

Parameters

<i>thee</i>	The Vfetk object
-------------	------------------

Definition at line [515](#) of file [vfetk.c](#).

Here is the caller graph for this function:

**8.2.3.17 VEXTERNC Vpbe* Vfetk_getVpbe (Vfetk * *thee*)**

Get a pointer to the Vpbe (PBE manager) object.

Author

Nathan Baker

Returns

Pointer to the Vpbe (PBE manager) object

Parameters

<i>thee</i>	The Vfetk object
-------------	------------------

Definition at line [508](#) of file [vfetk.c](#).

8.2.3.18 VEXTERNC Vrc_Codes Vfetk_loadGem (*Vfetk * thee, Gem * gm*)

Load a Gem geometry manager object into Vfetk.

Author

Nathan Baker

Parameters

<i>thee</i>	Destination
<i>gm</i>	Geometry manager source

8.2.3.19 VEXTERNC Vrc_Codes Vfetk_loadMesh (*Vfetk * thee, double center[3], double length[3], Vfetk_MeshLoad meshType, Vio * sock*)

Loads a mesh into the Vfetk (and associated) object(s).

Author

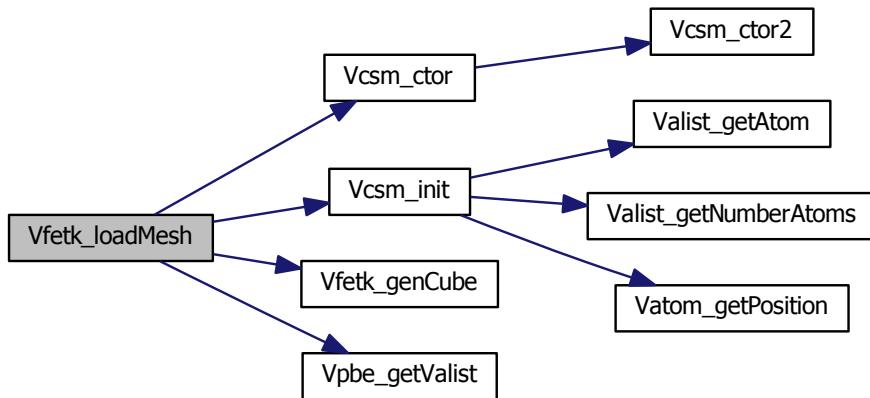
Nathan Baker

Parameters

<i>thee</i>	Vfetk object to load into
<i>center</i>	Center for mesh (if constructed)
<i>length</i>	Mesh lengths (if constructed)
<i>meshType</i>	Type of mesh to load
<i>sock</i>	Socket for external mesh data (NULL otherwise)

Definition at line 919 of file [vfetk.c](#).

Here is the call graph for this function:



8.2.3.20 VEXTERNC unsigned long int Vfetk_memChk (`Vfetk * thee`)

Return the memory used by this structure (and its contents) in bytes.

Author

Nathan Baker

Returns

The memory used by this structure and its contents in bytes

Parameters

<code>thee</code>	THe Vfetk object
-------------------	------------------

Definition at line 826 of file [vfetk.c](#).

Here is the call graph for this function:



8.2.3.21 `VEXTERNC void Vfetk_PDE_bisectEdge (int dim, int dimll, int edgeType, int chart[], double vx[]/[VAPBS_DIM])`

Define the way manifold edges are bisected.

Author

Nathan Baker and Mike Holst

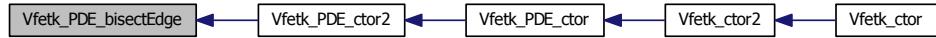
Note

This function is thread-safe.

Parameters

<code>dim</code>	Intrinsic dimension of manifold
<code>dimll</code>	Embedding dimension of manifold
<code>edgeType</code>	Type of edge being refined
<code>chart</code>	Chart for edge vertices, used here as accessibility bitfields
<code>vx</code>	Edge vertex coordinates

Here is the caller graph for this function:



8.2.3.22 VEXTERNC PDE* Vfetk_PDE_ctor (Vfetk * *fetk*)

Constructs the FEtk PDE object.

Author

Nathan Baker

Returns

Newly-allocated PDE object

Bug

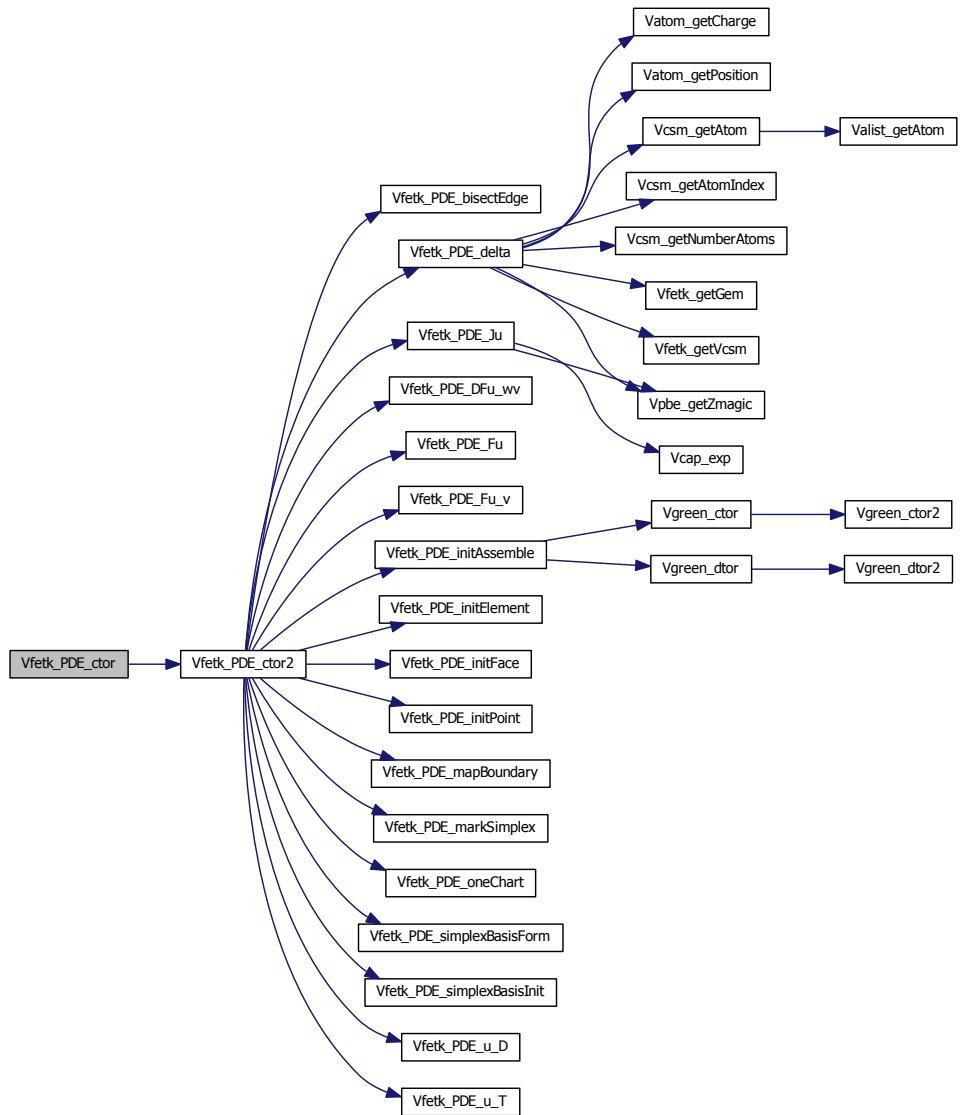
Not thread-safe

Parameters

<i>fetk</i>	The Vfetk object
-------------	------------------

Definition at line 1114 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3.23 VEXTERNC int Vfetk_PDE_ctor2 (PDE * *thee*, Vfetk * *fetk*)

Initializes the FEtk PDE object.

Author

Nathan Baker (with code by Mike Holst)

Returns

1 if successful, 0 otherwise

Bug

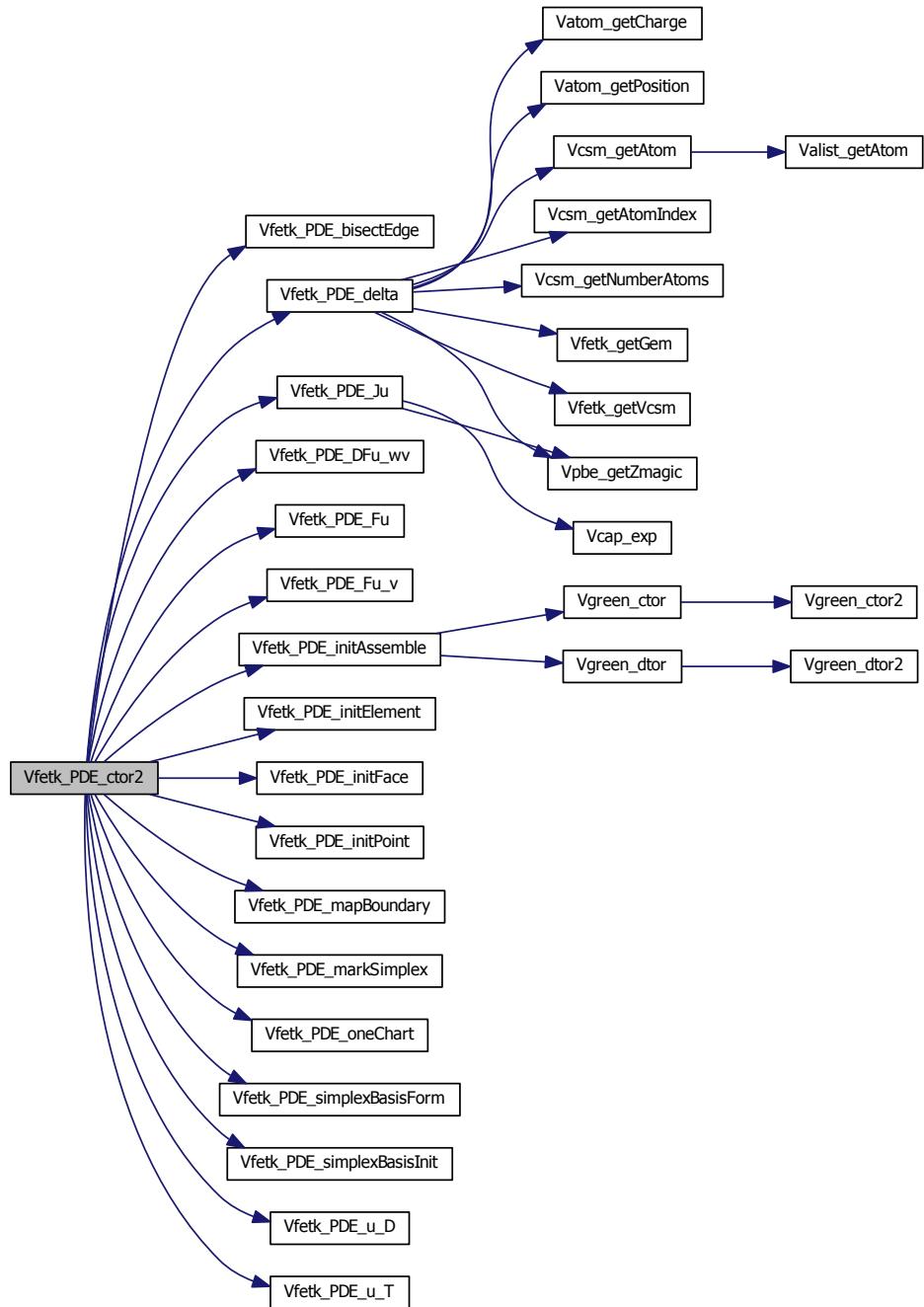
Not thread-safe

Parameters

<i>thee</i>	The newly-allocated PDE object
<i>fetk</i>	The parent Vfetk object

Definition at line 1125 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3.24 VEXTERNC void Vfetk_PDE_delta (PDE * *thee*, int *type*, int *chart*, double *txq[]*, void * *user*, double *F[]*)

Evaluate a (discretized) delta function source term at the given point.

Author

Nathan Baker

Bug

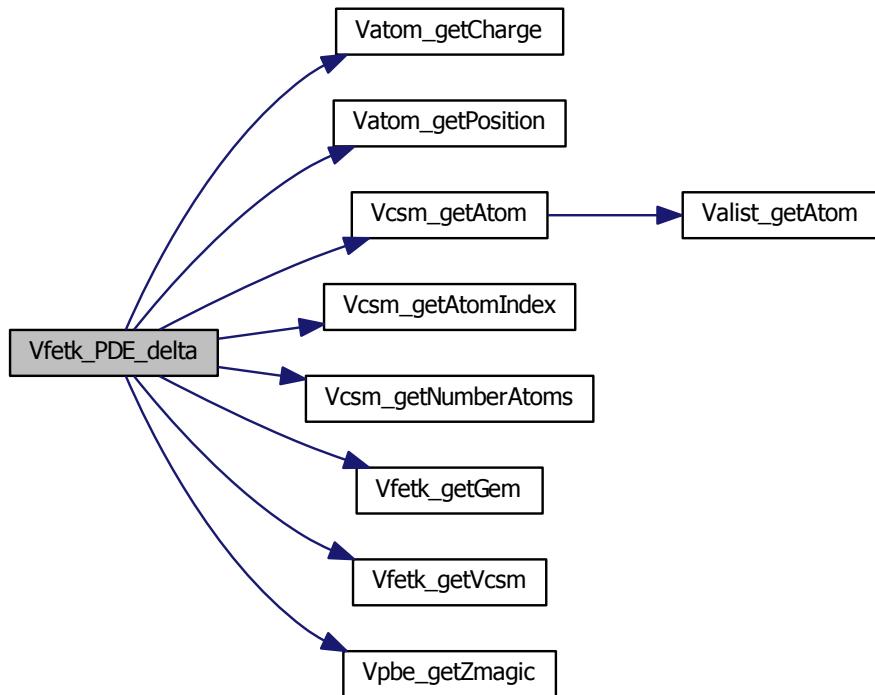
This function is not thread-safe

Parameters

<i>thee</i>	PDE object
<i>type</i>	Vertex type
<i>chart</i>	Chart for point coordinates
<i>txq</i>	Point coordinates
<i>user</i>	Vertex object pointer
<i>F</i>	Set to delta function value

Definition at line 1716 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3.25 VEXTERNC double Vfetk_PDE_DFu_wv (PDE * *thee*, int *key*, double *W*[], double *dW*[][VAPBS_DIM], double *V*[], double *dV*[][VAPBS_DIM])

This is the linearization of the weak form of the PBE; e.g., for use in a Newton iteration. This is the functional linearization of the strong form integrated with a test function to give:

$$\int_{\Omega} [\epsilon \nabla w \cdot \nabla v + b'(u) w v - f v] dx$$

where $b'(u)$ denotes the functional derivation of the mobile ion term.

Author

Nathan Baker and Mike Holst

Returns

Integrand value

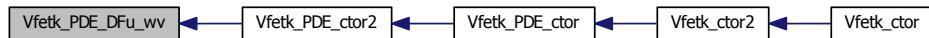
Bug

This function is not thread-safe

Parameters

<i>thee</i>	The PDE object
<i>key</i>	Integrand to evaluate (0 = interior weak form, 1 = boundary weak form)
<i>W</i>	Trial function value at current point
<i>dW</i>	Trial function gradient at current point
<i>V</i>	Test function value at current point
<i>dV</i>	Test function gradient

Here is the caller graph for this function:



8.2.3.26 VEXTERNC void Vfetk_PDE_dtor (PDE ** *thee*)

Destroys FEtk PDE object.

Author

Nathan Baker

Note

Thread-safe

Parameters

<i>thee</i>	Pointer to PDE object memory
-------------	------------------------------

Definition at line 1167 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**8.2.3.27 VEXTERNC void Vfetk_PDE_dtor2 (PDE * *thee*)**

FORTRAN stub: destroys FEtk PDE object.

Author

Nathan Baker

Note

Thread-safe

Parameters

<i>thee</i>	PDE object memory
-------------	-------------------

Definition at line 1182 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.28 VEXTERNC void Vfetk_PDE_Fu (PDE * *thee*, int *key*, double *F*[])

Evaluate strong form of PBE. For interior points, this is:

$$-\nabla \cdot \epsilon \nabla u + b(u) - f$$

where $b(u)$ is the (possibly nonlinear) mobile ion term and f is the source charge distribution term (for PBE) or the induced surface charge distribution (for RPBE). For an interior-boundary (simplex face) point, this is:

$$[\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^+} - [\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^-}$$

where $n(x)$ is the normal to the simplex face and the term represents the jump in dielectric displacement across the face. There is no outer-boundary contribution for this problem.

Author

Nathan Baker

Bug

This function is not thread-safe

This function is not implemented (sets error to zero)

Parameters

<i>thee</i>	The PDE object
<i>key</i>	Type of point (0 = interior, 1 = boundary, 2 = interior boundary)
<i>F</i>	Set to value of residual

Definition at line 1631 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.29 VEXTERNC double `Vfetk_PDE_Fu.v` (`PDE * thee, int key, double V[], double dV[] [VAPBS_DIM]`)

This is the weak form of the PBE; i.e. the strong form integrated with a test function to give:

$$\int_{\Omega} [\varepsilon \nabla u \cdot \nabla v + b(u)v - fv] dx$$

where $b(u)$ denotes the mobile ion term.

Author

Nathan Baker and Mike Holst

Returns

Integrand value

Bug

This function is not thread-safe

Parameters

<i>thee</i>	The PDE object
<i>key</i>	Integrand to evaluate (0 = interior weak form, 1 = boundary weak form)
<i>V</i>	Test function at current point
<i>dV</i>	Test function derivative at current point

Definition at line 1639 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.30 VEXTERNC void Vfetk_PDE_initAssemble (PDE * *thee*, int *ip*[], double *rp*[])

Do once-per-assembly initialization.

Author

Nathan Baker and Mike Holst

Note

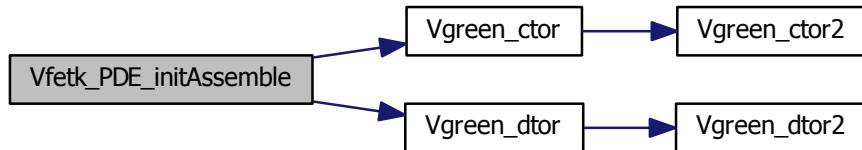
Thread-safe

Parameters

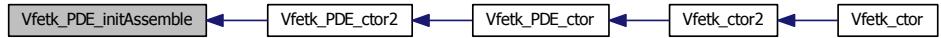
<i>thee</i>	PDE object
<i>ip</i>	Integer parameter array (not used)
<i>rp</i>	Double parameter array (not used)

Definition at line 1444 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3.31 VEXTERNC void Vfetk_PDE_initElement (PDE * *thee*, int *elementType*, int *chart*,
double *tvx*[][VAPBS_DIM], void * *data*)

Do once-per-element initialization.

Author

Nathan Baker and Mike Holst

Todo

Jump term is not implemented

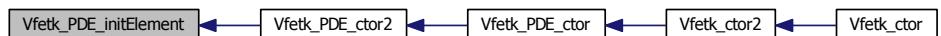
Bug

This function is not thread-safe

Parameters

<i>thee</i>	PDE object
<i>elementType</i>	Material type (not used)
<i>chart</i>	Chart in which the vertex coordinates are provided, used here as a bitfield to store molecular accessibility
<i>tvx</i>	Vertex coordinates
<i>data</i>	Simplex pointer (hack)

Here is the caller graph for this function:



```
8.2.3.32 VEXTERNC void Vfetk_PDE_initFace ( PDE * thee, int faceType, int chart, double
tnvec[] )
```

Do once-per-face initialization.

Author

Nathan Baker and Mike Holst

Bug

This function is not thread-safe

Parameters

<i>thee</i>	The PDE object
<i>faceType</i>	Simplex face type (interior or various boundary types)
<i>chart</i>	Chart in which the vertex coordinates are provided, used here as a bitfield for molecular accessibility
<i>tnvec</i>	Coordinates of outward normal vector for face

Definition at line 1495 of file [vfetk.c](#).

Here is the caller graph for this function:



```
8.2.3.33 VEXTERNC void Vfetk_PDE_initPoint ( PDE * thee, int pointType, int chart, double
txq[], double tU[], double tdU[][][VAPBS_DIM] )
```

Do once-per-point initialization.

Author

Nathan Baker

Bug

This function is not thread-safe

This function uses pre-defined boundary definitions for the molecular surface.

Parameters

<i>thee</i>	The PDE object
<i>pointType</i>	The type of point -- interior or various faces
<i>chart</i>	The chart in which the point coordinates are provided, used here as bitfield for molecular accessibility
<i>txq</i>	Point coordinates
<i>tU</i>	Solution value at point
<i>tdU</i>	Solution derivative at point

Here is the caller graph for this function:



8.2.3.34 VEXTERNC double Vfetk_PDE_Ju (PDE * *thee*, int *key*)

Energy functional. This returns the energy (less delta function terms) in the form:

$$c^{-1}/2 \int (\varepsilon(\nabla u)^2 + \kappa^2(cosh u - 1)) dx$$

for a 1:1 electrolyte where *c* is the output from Vpbe_getZmagic.

Author

Nathan Baker

Returns

Energy value (in kT)

Bug

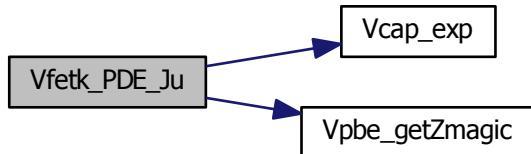
This function is not thread-safe.

Parameters

<i>thee</i>	The PDE object
<i>key</i>	What to evaluate: interior (0) or boundary (1)?

Definition at line 1926 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3.35 VEXTERNC void Vfetk_PDE_mapBoundary (int *dim*, int *dimll*, int *vertexType*, int *chart*, double *vx*[*VAPBS_DIM*])

Map a boundary point to some pre-defined shape.

Author

Nathan Baker and Mike Holst

Note

This function is thread-safe and is a no-op

Parameters

<i>dim</i>	Intrinsic dimension of manifold
<i>dimll</i>	Embedding dimension of manifold
<i>vertexType</i>	Type of vertex
<i>chart</i>	Chart for vertex coordinates
<i>vx</i>	Vertex coordinates

Here is the caller graph for this function:



8.2.3.36 `VEXTERNC int Vfetk_PDE_markSimplex (int dim, int dimll, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[]/[VAPBS_DIM], void * simplex)`

User-defined error estimator -- in our case, a geometry-based refinement method; forcing simplex refinement at the dielectric boundary and (for non-regularized PBE) the charges.

Author

Nathan Baker

Returns

1 if mark simplex for refinement, 0 otherwise

Bug

This function is not thread-safe

Parameters

<code>dim</code>	Intrinsic manifold dimension
<code>dimll</code>	Embedding manifold dimension
<code>simplexType</code>	Type of simplex being refined
<code>faceType</code>	Types of faces in simplex
<code>vertexType</code>	Types of vertices in simplex
<code>chart</code>	Charts for vertex coordinates
<code>vx</code>	Vertex coordinates
<code>simplex</code>	Simplex pointer

Here is the caller graph for this function:



8.2.3.37 VEXTERNC void Vfetk_PDE_oneChart (int *dim*, int *dimII*, int *objType*, int *chart[]*, double *vx[][][VAPBS_DIM]*, int *dimV*)

Unify the chart for different coordinate systems -- a no-op for us.

Author

Nathan Baker

Note

Thread-safe; a no-op

Parameters

<i>dim</i>	Intrinsic manifold dimension
<i>dimII</i>	Embedding manifold dimension
<i>objType</i>	???
<i>chart</i>	Charts of vertices' coordinates
<i>vx</i>	Vertices' coordinates
<i>dimV</i>	Number of vertices

Here is the caller graph for this function:



8.2.3.38 VEXTERNC void Vfetk_PDE_simplexBasisForm (int *key*, int *dim*, int *comp*, int *pdkey*, double *xq[]*, double *basis[]*)

Evaluate the bases for the trial or test space, for a particular component of the system, at all quadrature points on the master simplex element.

Author

Mike Holst

Parameters

<i>key</i>	Basis type to evaluate (0 = trial, 1 = test, 2 = trialB, 3 = testB)
<i>dim</i>	Spatial dimension
<i>comp</i>	Which component of elliptic system to produce basis for
<i>pdkey</i>	Basis partial differential equation evaluation key: <ul style="list-style-type: none">• 0 = evaluate basis(x,y,z)• 1 = evaluate basis_x(x,y,z)• 2 = evaluate basis_y(x,y,z)• 3 = evaluate basis_z(x,y,z)• 4 = evaluate basis_xx(x,y,z)• 5 = evaluate basis_yy(x,y,z)• 6 = evaluate basis_zz(x,y,z)• 7 = etc...
<i>xq</i>	Set to quad pt coordinate
<i>basis</i>	Set to all basis functions evaluated at all quadrature pts

Definition at line 2126 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.39 VEXTERNC int Vfetk_PDE_simplexBasisInit (int *key*, int *dim*, int *comp*, int * *ndof*, int *dof[]*)

Initialize the bases for the trial or the test space, for a particular component of the system, at all quadrature points on the master simplex element.

Author

Mike Holst

Note

```

*   The basis ordering is important. For a fixed quadrature
* point iq, you must follow the following ordering in p[iq][],
* based on how you specify the degrees of freedom in dof[]:
*
*   <v_0 vDF_0>,      <v_1 vDF_0>,      ..., <v_{nv} vDF_0>
*   <v_0 vDF_1>,      <v_1 vDF_1>,      ..., <v_{nv} vDF_1>
*   ...
*   <v_0 vDF_{nvDF}>, <v_0 vDF_{nvDF}>, ..., <v_{nv} vDF_{nvDF}>
*
*   <e_0 eDF_0>,      <e_1 eDF_0>,      ..., <e_{ne} eDF_0>
*   <e_0 eDF_1>,      <e_1 eDF_1>,      ..., <e_{ne} eDF_1>
*   ...
*   <e_0 eDF_{neDF}>, <e_1 eDF_{neDF}>, ..., <e_{ne} eDF_{neDF}>
*
*   <f_0 fDF_0>,      <f_1 fDF_0>,      ..., <f_{nf} fDF_0>
*   <f_0 fDF_1>,      <f_1 fDF_1>,      ..., <f_{nf} fDF_1>
*   ...
*   <f_0 fDF_{nfDF}>, <f_1 fDF_{nfDF}>, ..., <f_{nf} fDF_{nfDF}>
*
*   <s_0 sDF_0>,      <s_1 sDF_0>,      ..., <s_{ns} sDF_0>
*   <s_0 sDF_1>,      <s_1 sDF_1>,      ..., <s_{ns} sDF_1>
*   ...
*   <s_0 sDF_{nsDF}>, <s_1 sDF_{nsDF}>, ..., <s_{ns} sDF_{nsDF}>
*
*   For example, linear elements in R^3, with one degree of freedom at each *
* vertex, would use the following ordering:
*
*   <v_0 vDF_0>, <v_1 vDF_0>, <v_2 vDF_0>, <v_3 vDF_0>
*
*   Quadratic elements in R^2, with one degree of freedom at each vertex and
* edge, would use the following ordering:
*
*   <v_0 vDF_0>, <v_1 vDF_0>, <v_2 vDF_0>
*   <e_0 eDF_0>, <e_1 eDF_0>, <e_2 eDF_0>
*
*   You can use different trial and test spaces for each component of the
* elliptic system, thereby allowing for the use of Petrov-Galerkin methods.
* You MUST then tag the bilinear form symmetry entries as nonsymmetric in
* your PDE constructor to reflect that DF(u)(w,v) will be different from
* DF(u)(v,w), even if your form acts symmetrically when the same basis is
* used for w and v.
*
*   You can also use different trial spaces for each component of the elliptic
* system, and different test spaces for each component of the elliptic
* system. This allows you to e.g. use a basis which is vertex-based for
* one component, and a basis which is edge-based for another. This is
* useful in fluid mechanics, electromagnetics, or simply to play around with
* different elements.
*
*   This function is called by MC to build new master elements whenever it
* reads in a new mesh. Therefore, this function does not have to be all
* that fast, and e.g. could involve symbolic computation.

```

*

Parameters

<i>key</i>	Basis type to evaluate (0 = trial, 1 = test, 2 = trialB, 3 = testB)
<i>dim</i>	Spatial dimension
<i>comp</i>	Which component of elliptic system to produce basis for?
<i>ndof</i>	Set to the number of degrees of freedom
<i>dof</i>	Set to degree of freedom per v/e/f/s

Definition at line 2063 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.40 `VEXTERNC void Vfetk_PDE_u_D (PDE * thee, int type, int chart, double txq[], double F[])`

Evaluate the Dirichlet boundary condition at the given point.

Author

Nathan Baker

Bug

This function is hard-coded to call only multiple-sphere Debye-Hü functions.
This function is not thread-safe.

Parameters

<i>thee</i>	PDE object
<i>type</i>	Vertex boundary type
<i>chart</i>	Chart for point coordinates
<i>txq</i>	Point coordinates
<i>F</i>	Set to boundary values

Definition at line 1803 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.41 VEXTERNC void Vfetk_PDE_u_T (PDE * *thee*, int *type*, int *chart*, double *txq*[], double *F*[])

Evaluate the "true solution" at the given point for comparison with the numerical solution.

Author

Nathan Baker

Note

This function only returns zero.

Bug

This function is not thread-safe.

Parameters

<i>thee</i>	PDE object
<i>type</i>	Point type
<i>chart</i>	Chart for point coordinates
<i>txq</i>	Point coordinates
<i>F</i>	Set to value at point

Definition at line 1816 of file [vfetk.c](#).

Here is the caller graph for this function:



8.2.3.42 VEXTERNC double Vfetk_qfEnergy (*Vfetk * thee*, int *color*)

Get the "fixed charge" contribution to the electrostatic energy.

Using the solution at the finest mesh level, get the electrostatic energy due to the interaction of the fixed charges with the potential:

$$G = \sum_i q_i u(r_i)$$

and return the result in units of $k_B T$. Clearly, no self-interaction terms are removed. A factor a 1/2 has to be included to convert this to a real energy.

Author

Nathan Baker

Parameters

<i>thee</i>	Vfetk object
<i>color</i>	Partition restriction for energy evaluation, only used if non-negative

Returns

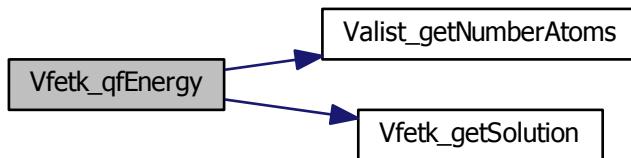
The fixed charge electrostatic energy in units of $k_B T$.

Parameters

<i>thee</i>	The Vfetk object
<i>color</i>	Partition restriction for energy evaluation, only used if non-negative

Definition at line 700 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3.43 VEXTERNC void Vfetk.readMesh (*Vfetk * thee, int skey, Vio * sock*)

Read in mesh and initialize associated internal structures.

Author

Nathan Baker

Note

See also

[Vfetk_genCube](#)

Parameters

<i>thee</i>	THe Vfetk object
<i>skey</i>	The sock format key (0 = MCSF simplex format)
<i>sock</i>	Socket object ready for reading

8.2.3.44 VEXTERNC void Vfetk.setAtomColors (*Vfetk * thee*)

Transfer color (partition ID) information frmo a partitioned mesh to the atoms.

Transfer color information from partitioned mesh to the atoms. In the case that a charge is shared between two partitions, the partition color of the first simplex is selected. Due to the arbitrary nature of this selection, THIS METHOD SHOULD ONLY BE USED IMMEDIATELY AFTER PARTITIONING!!!

Warning

This function should only be used immediately after mesh partitioning

Author

Nathan Baker

Note

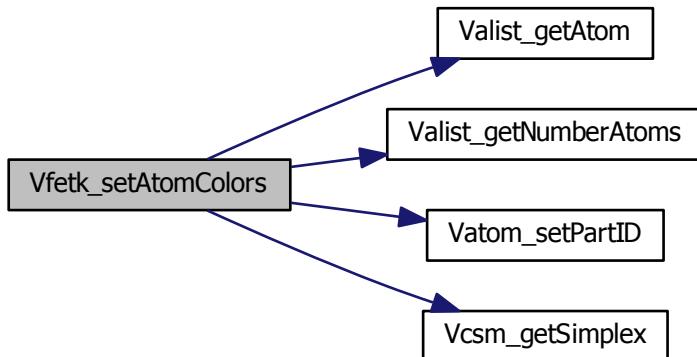
This is a friend function of Vcsm

Parameters

<i>thee</i>	THe Vfetk object
-------------	------------------

Definition at line 808 of file [vfetk.c](#).

Here is the call graph for this function:



8.2.3.45 VEXTERNC void Vfetk_setParameters (**Vfetk** * *thee*, **PBEparm** * *pbeparm*, **FEMparm** * *feparm*)

Set the parameter objects.

Author

Nathan Baker

Parameters

<i>thee</i>	The Vfetk object
<i>pbeparm</i>	Parameters for solution of the PBE
<i>feparm</i>	FEM-specific solution parameters

Definition at line 613 of file [vfetk.c](#).

8.2.3.46 VEXTERNC int Vfetk_write (Vfetk * *thee*, const char * *iodev*, const char * *iofmt*, const char * *thost*, const char * *fname*, Bvec * *vec*, Vdata_Format *format*)

Write out data.

Author

Nathan Baker

Parameters

<i>thee</i>	Vfetk object
<i>vec</i>	FEtk Bvec vector to use
<i>format</i>	Format for data
<i>iodev</i>	Output device type (FILE/BUFF/UNIX/INET)
<i>iofmt</i>	Output device format (ASCII/XDR)
<i>thost</i>	Output hostname (for sockets)
<i>fname</i>	Output FILE/BUFF/UNIX/INET name

Note

This function is thread-safe

Bug

Some values of format are not implemented

Returns

1 if successful, 0 otherwise

Parameters

<i>thee</i>	The Vfetk object
<i>iodev</i>	Output device type (FILE = file, BUFF = buffer, UNIX = unix pipe, INET = network socket)
<i>iofmt</i>	Output device format (ASCII = ascii/plaintext, XDR = xdr)
<i>thost</i>	Output hostname for sockets
<i>fname</i>	Output filename for other
<i>vec</i>	Data vector
<i>format</i>	Data format

Definition at line 2387 of file [vfetk.c](#).

8.3 Vpee class

This class provides some functionality for error estimation in parallel.

Data Structures

- struct [sVpee](#)

Contains public data members for Vpee class/module.

Files

- file [vpee.h](#)

Contains declarations for class Vpee.

- file [vpee.c](#)

Class Vpee methods.

Typedefs

- typedef struct [sVpee Vpee](#)

Declaration of the Vpee class as the Vpee structure.

Functions

- VEXTERNC [Vpee * Vpee_ctor](#) (Gem *gm, int localPartID, int killFlag, double killParam)

Construct the Vpee object.

- VEXTERNC int [Vpee_ctor2 \(Vpee *thee, Gem *gm, int localPartID, int killFlag, double killParam\)](#)

FORTRAN stub to construct the Vpee object.

- VEXTERNC void [Vpee_dtor \(Vpee **thee\)](#)

Object destructor.

- VEXTERNC void [Vpee_dtor2 \(Vpee *thee\)](#)

FORTRAN stub object destructor.

- VEXTERNC int [Vpee_markRefine \(Vpee *thee, AM *am, int level, int akey, int rcol, double etol, int bkey\)](#)

Mark simplices for refinement based on attenuated error estimates.

- VEXTERNC int **Vpee_numSS** (**Vpee** **thee*)
Returns the number of simplices in the local partition.

8.3.1 Detailed Description

This class provides some functionality for error estimation in parallel. This class provides some functionality for error estimation in parallel. The purpose is to modulate the error returned by some external error estimator according to the partitioning of the mesh. For example, the Bank/Holst parallel refinement routine essentially reduces the error outside the "local" partition to zero. However, this leads to the need for a few final overlapping Schwarz solves to smooth out the errors near partition boundaries. Supposedly, if the region in which we allow error-based refinement includes the "local" partition and an external buffer zone approximately equal in size to the local region, then the solution will asymptotically approach the solution obtained via more typical methods. This is essentially a more flexible parallel implementation of MC's AM_markRefine.

8.3.2 Function Documentation

8.3.2.1 VEXTERNC **Vpee*** **Vpee_ctor** (**Gem** **gm*, int *localPartID*, int *killFlag*, double *killParam*)

Construct the Vpee object.

Author

Nathan Baker

Returns

Newly constructed Vpee object

Parameters

<i>gm</i>	FEtk geometry manager object
<i>localPartID</i>	ID of the local partition (focus of refinement)
<i>killFlag</i>	A flag to indicate how error estimates are to be attenuated outside the local partition: <ul style="list-style-type: none"> • 0: no attenuation • 1: all error outside the local partition set to zero • 2: all error is set to zero outside a sphere of radius (<i>killParam</i>*<i>partRadius</i>), where <i>partRadius</i> is the radius of the sphere circumscribing the local partition • 3: all error is set to zero except for the local partition and its immediate neighbors
<i>killParam</i>	

See also

[killFlag](#) for usage

Definition at line 84 of file [vpee.c](#).

8.3.2.2 VEXTERNC int Vpee_ctor2 (Vpee * *thee*, Gem * *gm*, int *localPartID*, int *killFlag*, double *killParam*)

FORTRAN stub to construct the Vpee object.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

<i>thee</i>	The Vpee object
<i>gm</i>	FEtk geometry manager object
<i>localPartID</i>	ID of the local partition (focus of refinement)
<i>killFlag</i>	A flag to indicate how error estimates are to be attenuated outside the local partition: <ul style="list-style-type: none"> • 0: no attenuation • 1: all error outside the local partition set to zero • 2: all error is set to zero outside a sphere of radius (<i>killParam</i>*partRadius), where partRadius is the radius of the sphere circumscribing the local partition • 3: all error is set to zero except for the local partition and its immediate neighbors
<i>killParam</i>	

See also

[killFlag](#) for usage

Definition at line 102 of file [vpee.c](#).

8.3.2.3 VEXTERNC void Vpee_dtor (Vpee ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of the Vpee object
-------------	---

Definition at line [205](#) of file [vpee.c](#).

8.3.2.4 VEXTERNC void Vpee_dtor2 (Vpee * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
-------------	-----------------------------------

Definition at line [220](#) of file [vpee.c](#).

8.3.2.5 VEXTERNC int Vpee_markRefine (Vpee * *thee*, AM * *am*, int *level*, int *akey*, int *rcol*, double *etol*, int *bkey*)

Mark simplices for refinement based on attenuated error estimates.

A wrapper/reimplementation of AM_markRefine that allows for more flexible attenuation of error-based markings outside the local partition. The error in each simplex is modified by the method (see killFlag) specified in the Vpee constructor. This allows the user to confine refinement to an arbitrary area around the local partition.

Author

Nathan Baker and Mike Holst

Note

This routine borrows very heavily from FETk routines by Mike Holst.

Returns

The number of simplices marked for refinement.

Bug

This function is no longer up-to-date with FETk and may not function properly

Parameters

<i>thee</i>	The Vpee object
<i>am</i>	The FETk algebra manager currently used to solve the PB
<i>level</i>	The current level of the multigrid hierarchy
<i>akey</i>	<p>The marking method:</p> <ul style="list-style-type: none"> • -1: Reset markings --> killFlag has no effect. • 0: Uniform. • 1: User defined (geometry-based). • >1: A numerical estimate for the error has already been set in am and should be attenuated according to killFlag and used, in conjunction with etol, to mark simplices for refinement.
<i>rcol</i>	The ID of the main partition on which to mark (or -1 if all partitions should be marked). Note that we should have (<i>rcol</i> == <i>thee</i> ->localPartID) for (<i>thee</i> ->killFlag == 2 or 3)
<i>etol</i>	The error tolerance criterion for marking
<i>bkey</i>	<p>How the error tolerance is interpreted:</p> <ul style="list-style-type: none"> • 0: Simplex marked if error > etol. • 1: Simplex marked if error > $\sqrt{etol^2/L}$ where L\$ is the number of simplices

Definition at line 228 of file [vpee.c](#).

8.3.2.6 VEXTERNC int Vpee_numSS (Vpee * *thee*)

Returns the number of simplices in the local partition.

Author

Nathan Baker

Returns

Number of simplices in the local partition

Parameters

<i>thee</i>	The Vpee object
-------------	-----------------

Definition at line 446 of file [vpee.c](#).

8.4 APOLparm class

Parameter structure for APOL-specific variables from input files.

Collaboration diagram for APOLparm class:



Data Structures

- struct [sAPOLparm](#)

Parameter structure for APOL-specific variables from input files.

Files

- file [femparm.h](#)

Contains declarations for class APOLparm.

- file [apolparm.c](#)

Class APOLparm methods.

Typedefs

- typedef enum [eAPOLparm_calcEnergy](#) APOLparm_calcEnergy

Define eAPOLparm_calcEnergy enumeration as APOLparm_calcEnergy.

- typedef enum [eAPOLparm_calcForce](#) APOLparm_calcForce

Define eAPOLparm_calcForce enumeration as APOLparm_calcForce.

- typedef enum [eAPOLparm_doCalc](#) APOLparm_doCalc

Define eAPOLparm_calcForce enumeration as APOLparm_calcForce.

- typedef struct [sAPOLparm](#) APOLparm

Declaration of the APOLparm class as the APOLparm structure.

Enumerations

- enum eAPOLparm_calcEnergy { ACE_NO = 0, ACE_TOTAL = 1, ACE_COMPS = 2 }
Define energy calculation enumeration.
- enum eAPOLparm_calcForce { ACF_NO = 0, ACF_TOTAL = 1, ACF_COMPS = 2 }
Define force calculation enumeration.
- enum eAPOLparm_doCalc { ACD_NO = 0, ACD_YES = 1, ACD_ERROR = 2 }
Define force calculation enumeration.

Functions

- VEXTERNC APOLparm * APOLparm_ctor ()
Construct APOLparm.
- VEXTERNC Vrc_Codes APOLparm_ctor2 (APOLparm *thee)
FORTRAN stub to construct APOLparm.
- VEXTERNC void APOLparm_dtor (APOLparm **thee)
Object destructor.
- VEXTERNC void APOLparm_dtor2 (APOLparm *thee)
FORTRAN stub for object destructor.
- VEXTERNC Vrc_Codes APOLparm_check (APOLparm *thee)
Consistency check for parameter values stored in object.
- VEXTERNC void APOLparm_copy (APOLparm *thee, APOLparm *source)
Copy target object into thee.

8.4.1 Detailed Description

Parameter structure for APOL-specific variables from input files.

8.4.2 Enumeration Type Documentation

8.4.2.1 enum eAPOLparm_calcEnergy

Define energy calculation enumeration.

Enumerator:

- ACE_NO** Do not perform energy calculation
- ACE_TOTAL** Calculate total energy only

ACE_COMP Calculate per-atom energy components

Definition at line [76](#) of file [apolparm.h](#).

8.4.2.2 enum eAPOLparm_calcForce

Define force calculation enumeration.

Enumerator:

ACF_NO Do not perform force calculation

ACF_TOTAL Calculate total force only

ACF_COMP Calculate per-atom force components

Definition at line [92](#) of file [apolparm.h](#).

8.4.2.3 enum eAPOLparm_doCalc

Define force calculation enumeration.

Enumerator:

ACD_NO Do not perform calculation

ACD_YES Perform calculations

ACD_ERROR Error setting up calculation

Definition at line [108](#) of file [apolparm.h](#).

8.4.3 Function Documentation

8.4.3.1 VEXTERNC Vrc_Codes APOLparm_check (APOLparm * *thee*)

Consistency check for parameter values stored in object.

Author

David Gohara, Yong Huang

Parameters

<i>thee</i>	APOLparm object
-------------	-----------------

Returns

Success enumeration

Definition at line 181 of file [apolparm.c](#).

8.4.3.2 VEXTERNC void APOLparm_copy (APOLparm * *thee*, APOLparm * *source*)

Copy target object into thee.

Author

Nathan Baker

Parameters

<i>thee</i>	Destination object
<i>source</i>	Source object

Definition at line 110 of file [apolparm.c](#).

Here is the caller graph for this function:

**8.4.3.3 VEXTERNC APOLparm* APOLparm_ctor ()**

Construct APOLparm.

Author

David Gohara

Returns

Newly allocated and initialized Vpmgp object

Definition at line 67 of file [apolparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.4 VEXTERNC Vrc_Codes APOLparm_ctor2(APOLparm * *thee*)

FORTRAN stub to construct APOLparm.

Author

David Gohara, Yong Huang

Parameters

<i>thee</i>	Pointer to allocated APOLparm object
-------------	--------------------------------------

Returns

Success enumeration

Definition at line [78](#) of file [apolparm.c](#).

Here is the caller graph for this function:



8.4.3.5 VEXTERNC void APOLparm_dtor (APOLparm ** *thee*)

Object destructor.

Author

David Gohara

Parameters

<i>thee</i>	Pointer to memory location of APOLparm object
-------------	---

Definition at line 169 of file [apolparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.6 VEXTERNC void APOLparm_dtor2 (APOLparm * *thee*)

FORTRAN stub for object destructor.

Author

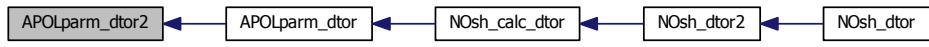
David Gohara

Parameters

<i>thee</i>	Pointer to APOLparm object
-------------	----------------------------

Definition at line 179 of file [apolparm.c](#).

Here is the caller graph for this function:



8.5 FEMparm class

Parameter structure for FEM-specific variables from input files.

Collaboration diagram for FEMparm class:



Data Structures

- struct [sFEMparm](#)

Parameter structure for FEM-specific variables from input files.

Files

- file [femparm.h](#)

Contains declarations for class APOLparm.

- file [femparm.c](#)

Class FEMparm methods.

Typedefs

- typedef enum [eFEMparm_EtolType](#) FEMparm_EtolType

Declare FEmperm_EtolType type.

- typedef enum [eFEMparm_EstType](#) FEMparm_EstType

Declare FEMperm_EstType type.

- typedef enum [eFEMparm_CalcType](#) FEMparm_CalcType

Declare FEMperm_CalcType type.

- typedef struct [sFEMparm](#) FEMparm

Declaration of the FEMparm class as the FEMparm structure.

Enumerations

- enum [eFEMparm_EtolType](#) { [FET_SIMP](#) = 0, [FET_GLOB](#) = 1, [FET_FRAC](#) = 2 }

Adaptive refinement error estimate tolerance key.

- enum eFEMparm_EstType {
 FRT_UNIF = 0, FRT_GEOM = 1, FRT_RESI = 2, FRT_DUAL = 3,
 FRT_LOCA = 4 }
Adaptive refinement error estimator method.
- enum eFEMparm_CalcType { FCT_MANUAL, FCT_NONE }
Calculation type.

Functions

- VEXTERNC FEMparm * FEMparm_ctor (FEMparm_CalcType type)
Construct FEMparm.
- VEXTERNC int FEMparm_ctor2 (FEMparm *thee, FEMparm_CalcType type)
FORTRAN stub to construct FEMparm.
- VEXTERNC void FEMparm_dtor (FEMparm **thee)
Object destructor.
- VEXTERNC void FEMparm_dtor2 (FEMparm *thee)
FORTRAN stub for object destructor.
- VEXTERNC int FEMparm_check (FEMparm *thee)
Consistency check for parameter values stored in object.
- VEXTERNC void FEMparm_copy (FEMparm *thee, FEMparm *source)
Copy target object into thee.

8.5.1 Detailed Description

Parameter structure for FEM-specific variables from input files.

8.5.2 Typedef Documentation

8.5.2.1 `typedef enum eFEMparm_EtolType FEMparm_EtolType`

Declare FEm parm_EtolType type.

Author

Nathan Baker

Definition at line 87 of file `femparm.h`.

8.5.3 Enumeration Type Documentation

8.5.3.1 enum eFEMparm_CalcType

Calculation type.

Enumerator:

FCT_MANUAL fe-manual

FCT_NONE unspecified

Definition at line 114 of file [femparm.h](#).

8.5.3.2 enum eFEMparm_EstType

Adaptive refinement error estimator method.

Note

Do not change these values; they correspond to settings in FEtk

Author

Nathan Baker

Enumerator:

FRT_UNIF Uniform refinement

FRT_GEOM Geometry-based (i.e. surfaces and charges) refinement

FRT_RESI Nonlinear residual estimate-based refinement

FRT_DUAL Dual-solution weight nonlinear residual estimate-based refinement

FRT_LOCA Local problem error estimate-based refinement

Definition at line 95 of file [femparm.h](#).

8.5.3.3 enum eFEMparm_EtolType

Adaptive refinement error estimate tolerance key.

Author

Nathan Baker

Enumerator:

FET_SIMP per-simplex error tolerance

FET_GLOB global error tolerance

FET_FRAC fraction of simplices we want to have refined

Definition at line [76](#) of file [femparm.h](#).

8.5.4 Function Documentation

8.5.4.1 VEXTERNC int FEMparm_check (FEMparm * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Parameters

<i>thee</i>	FEMparm object
-------------	----------------

Returns

1 if OK, 0 otherwise

Definition at line [143](#) of file [femparm.c](#).

8.5.4.2 VEXTERNC void FEMparm_copy (FEMparm * *thee*, FEMparm * *source*)

Copy target object into thee.

Author

Nathan Baker

Parameters

<i>thee</i>	Destination object
<i>source</i>	Source object

Definition at line [100](#) of file [femparm.c](#).

Here is the caller graph for this function:



8.5.4.3 VEXTERNC FEMparm* FEMparm_ctor (FEMparm_CalcType type)

Construct FEMparm.

Author

Nathan Baker

Parameters

type	FEM calculation type
------	----------------------

Returns

Newly allocated and initialized Vpmgp object

Definition at line 67 of file [femparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.4.4 VEXTERNC int FEMparm_ctor2 (FEMparm * *thee*, FEMparm_CalcType *type*)

FORTRAN stub to construct FEMparm.

Author

Nathan Baker

Parameters

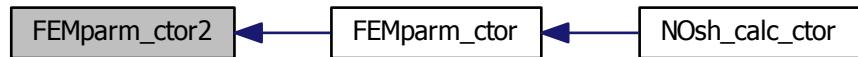
<i>thee</i>	Pointer to allocated FEMparm object
<i>type</i>	FEM calculation type

Returns

1 if successful, 0 otherwise

Definition at line 78 of file [femparm.c](#).

Here is the caller graph for this function:



8.5.4.5 VEXTERNC void FEMparm_dtor (FEMparm ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of FEMparm object
-------------	--

Definition at line 133 of file [femparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**8.5.4.6 VEXTERNC void FEMparm_dtor2 (FEMparm * *thee*)**

FORTRAN stub for object destructor.

Author

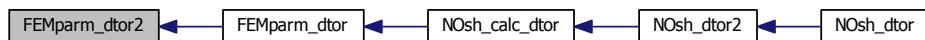
Nathan Baker

Parameters

<i>thee</i>	Pointer to FEMparm object
-------------	---------------------------

Definition at line 141 of file [femparm.c](#).

Here is the caller graph for this function:



8.6 MGparm class

Parameter which holds useful parameters for generic multigrid calculations.

Data Structures

- struct [sMGparm](#)

Parameter structure for MG-specific variables from input files.

Files

- file [mgparm.h](#)

Contains declarations for class MGparm.

- file [mgparm.c](#)

Class MGparm methods.

Typedefs

- typedef enum [eMGparm_CalcType](#) MGparm_CalcType

Declare MGparm_CalcType type.

- typedef enum [eMGparm_CentMeth](#) MGparm_CentMeth

Declare MGparm_CentMeth type.

- typedef struct [sMGparm](#) MGparm

Declaration of the MGparm class as the MGparm structure.

Enumerations

- enum eMGparm_CalcType {

 MCT_MANUAL = 0, **MCT_AUTO** = 1, **MCT_PARALLEL** = 2, **MCT_DUMMY** = 3,
MCT_NONE = 4 }

Calculation type.
- enum eMGparm_CentMeth { **MCM_POINT** = 0, **MCM_MOLECULE** = 1, **MCM_FOCUS** = 2 }

Centering method.

Functions

- VEXTERNC Vrc_Codes **APOLparm_parseToken** (**APOLparm** *thee, char tok[VMAX_-BUFSIZE], Vio *sock)

Parse an MG keyword from an input file.
- VEXTERNC Vrc_Codes **FEMparm_parseToken** (**FEMparm** *thee, char tok[VMAX_-BUFSIZE], Vio *sock)

Parse an MG keyword from an input file.
- VEXTERNC int **MGparm_getNx** (**MGparm** *thee)

Get number of grid points in x direction.
- VEXTERNC int **MGparm_getNy** (**MGparm** *thee)

Get number of grid points in y direction.
- VEXTERNC int **MGparm_getNz** (**MGparm** *thee)

Get number of grid points in z direction.
- VEXTERNC double **MGparm_getHx** (**MGparm** *thee)

Get grid spacing in x direction (Å)
- VEXTERNC double **MGparm_getHy** (**MGparm** *thee)

Get grid spacing in y direction (Å)
- VEXTERNC double **MGparm_getHz** (**MGparm** *thee)

Get grid spacing in z direction (Å)
- VEXTERNC void **MGparm_setCenterX** (**MGparm** *thee, double x)

Set center x-coordinate.
- VEXTERNC void **MGparm_setCenterY** (**MGparm** *thee, double y)

Set center y-coordinate.
- VEXTERNC void **MGparm_setCenterZ** (**MGparm** *thee, double z)

Set center z-coordinate.
- VEXTERNC double **MGparm_getCenterX** (**MGparm** *thee)

Get center x-coordinate.
- VEXTERNC double **MGparm_getCenterY** (**MGparm** *thee)

Get center y-coordinate.

- VEXTERNC double `MGparm_getCenterZ (MGparm *thee)`
Get center z-coordinate.
- VEXTERNC `MGparm * MGparm_ctor (MGparm_CalcType type)`
Construct MGparm object.
- VEXTERNC `Vrc_Codes MGparm_ctor2 (MGparm *thee, MGparm_CalcType type)`
FORTRAN stub to construct MGparm object.
- VEXTERNC void `MGparm_dtor (MGparm **thee)`
Object destructor.
- VEXTERNC void `MGparm_dtor2 (MGparm *thee)`
FORTRAN stub for object destructor.
- VEXTERNC `Vrc_Codes MGparm_check (MGparm *thee)`
Consistency check for parameter values stored in object.
- VEXTERNC void `MGparm_copy (MGparm *thee, MGparm *parm)`
Copy MGparm object into thee.
- VEXTERNC `Vrc_Codes MGparm_parseToken (MGparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)`
Parse an MG keyword from an input file.

8.6.1 Detailed Description

Parameter which holds useful parameters for generic multigrid calculations.

8.6.2 Enumeration Type Documentation

8.6.2.1 enum eMGparm_CalcType

Calculation type.

Enumerator:

- `MCT_MANUAL` mg-manual
- `MCT_AUTO` mg-auto
- `MCT_PARALLEL` mg-para
- `MCT_DUMMY` mg-dummy
- `MCT_NONE` unspecified

Definition at line 74 of file `mgparm.h`.

8.6.2.2 enum eMGparm_CentMeth

Centering method.

Enumerator:

- MCM_POINT** Center on a point
- MCM_MOLECULE** Center on a molecule
- MCM_FOCUS** Determined by focusing

Definition at line 92 of file [mgparm.h](#).

8.6.3 Function Documentation

8.6.3.1 VEXTERNC Vrc_Codes APOLparm_parseToken (APOLparm * *thee*, char *tok*[VMAX_BUFSIZE], Vio * *sock*)

Parse an MG keyword from an input file.

Author

David Gohara

Parameters

<i>thee</i>	MGparm object
<i>tok</i>	Token to parse
<i>sock</i>	Stream for more tokens

Returns

Success enumeration (1 if matched and assigned; -1 if matched, but there's some sort of error (i.e., too few args); 0 if not matched)

Definition at line 579 of file [apolparm.c](#).

Here is the call graph for this function:



8.6.3.2 VEXTERNC Vrc_Codes FEMparm_parseToken (FEMparm * *thee*, char *tok*[*VMAX_BUFSIZE*], Vio * *sock*)

Parse an MG keyword from an input file.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
<i>tok</i>	Token to parse
<i>sock</i>	Stream for more tokens

Returns

VRC_SUCCESS if matched and assigned; VRC_FAILURE if matched, but there's some sort of error (i.e., too few args); VRC_WARNING if not matched

Definition at line [425](#) of file [femparm.c](#).

Here is the call graph for this function:



8.6.3.3 VEXTERNC Vrc_Codes MGparm_check (MGparm * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

Success enumeration

Definition at line [189](#) of file [mgparm.c](#).

8.6.3.4 VEXTERNC void MGparm_copy (MGparm * *thee*, MGparm * *parm*)

Copy MGparm object into thee.

Author

Nathan Baker and Todd Dolinsky

Parameters

<i>thee</i>	MGparm object (target for copy)
<i>parm</i>	MGparm object (source for copy)

Definition at line [345](#) of file [mgparm.c](#).

Here is the caller graph for this function:



8.6.3.5 VEXTERNC MGparm* MGparm_ctor (MGparm_CalcType type)

Construct MGparm object.

Author

Nathan Baker

Parameters

<i>type</i>	Type of MG calculation
-------------	------------------------

Returns

Newly allocated and initialized MGparm object

Definition at line 118 of file [mgparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**8.6.3.6 VEXTERNC Vrc_Codes MGparm_ctor2 (MGparm * thee, MGparm_CalcType type)**

FORTRAN stub to construct MGparm object.

Author

Nathan Baker and Todd Dolinsky

Parameters

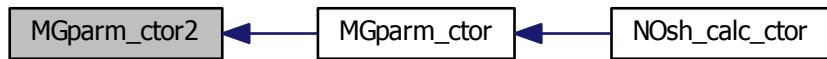
<i>thee</i>	Space for MGparm object
<i>type</i>	Type of MG calculation

Returns

Success enumeration

Definition at line [129](#) of file [mgparm.c](#).

Here is the caller graph for this function:

**8.6.3.7 VEXTERNC void MGparm_dtor(MGparm ** *thee*)**

Object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of MGparm object
-------------	---

Definition at line [179](#) of file [mgparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.6.3.8 VEXTERN void MGparm_dtor2 (MGparm * *thee*)

FORTRAN stub for object destructor.

Author

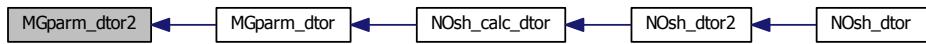
Nathan Baker

Parameters

<i>thee</i>	Pointer to MGparm object
-------------	--------------------------

Definition at line [187](#) of file [mgparm.c](#).

Here is the caller graph for this function:



8.6.3.9 VEXTERNC double MGparm_getCenterX (MGparm * *thee*)

Get center x-coordinate.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

x-coordinate

Definition at line 81 of file [mgparm.c](#).

8.6.3.10 VEXTERNC double MGparm_getCenterY (MGparm * *thee*)

Get center y-coordinate.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

y-coordinate

Definition at line 85 of file [mgparm.c](#).

8.6.3.11 VEXTERNC double MGparm_getCenterZ (MGparm * *thee*)

Get center z-coordinate.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

z-coordinate

Definition at line 89 of file [mgparm.c](#).

8.6.3.12 VEXTERNC double MGparm_getHx (MGparm * *thee*)

Get grid spacing in x direction (Å)

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

Grid spacing in the x direction

Definition at line 105 of file [mgparm.c](#).

8.6.3.13 VEXTERNC double MGparm_getHy (MGparm * *thee*)

Get grid spacing in y direction (Å)

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

Grid spacing in the y direction

Definition at line 109 of file [mgparm.c](#).

8.6.3.14 VEXTERNC double MGparm_getHz (MGparm * *thee*)

Get grid spacing in z direction (Å)

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

Grid spacing in the z direction

Definition at line 113 of file [mgparm.c](#).

8.6.3.15 VEXTERNC int MGparm_getNx (MGparm * *thee*)

Get number of grid points in x direction.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

Number of grid points in the x direction

Definition at line 93 of file [mgparm.c](#).

8.6.3.16 VEXTERNC int MGparm_getNy (MGparm * *thee*)

Get number of grid points in y direction.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

Number of grid points in the y direction

Definition at line [97](#) of file [mgparm.c](#).

8.6.3.17 VEXTERNC int MGparm_getNz (MGparm * *thee*)

Get number of grid points in z direction.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
-------------	---------------

Returns

Number of grid points in the z direction

Definition at line [101](#) of file [mgparm.c](#).

8.6.3.18 VEXTERNC Vrc_Codes MGparm_parseToken (MGparm * *thee*, char *tok*[VMAX_BUFSIZE], Vio * *sock*)

Parse an MG keyword from an input file.

Author

Nathan Baker and Todd Dolinsky

Parameters

<i>thee</i>	MGparm object
<i>tok</i>	Token to parse
<i>sock</i>	Stream for more tokens

Returns

Success enumeration (1 if matched and assigned; -1 if matched, but there's some sort of error (i.e., too few args); 0 if not matched)

Definition at line 923 of file [mgparm.c](#).

Here is the call graph for this function:



8.6.3.19 VEXTERNC void MGparm_setCenterX (MGparm * *thee*, double *x*)

Set center x-coordinate.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
<i>x</i>	x-coordinate

Definition at line 69 of file [mgparm.c](#).

8.6.3.20 VEXTERNC void MGparm_setCenterY (MGparm * *thee*, double *y*)

Set center y-coordinate.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
<i>y</i>	y-coordinate

Definition at line 73 of file [mgparm.c](#).

8.6.3.21 VEXTERNC void MGparm.setCenterZ(MGparm * *thee*, double *z*)

Set center z-coordinate.

Author

Nathan Baker

Parameters

<i>thee</i>	MGparm object
<i>z</i>	z-coordinate

Definition at line 77 of file [mgparm.c](#).

8.7 NOsh class

Class for parsing for fixed format input files.

Data Structures

- struct [sNOsh_calc](#)
Calculation class for use when parsing fixed format input files.
- struct [sNOsh](#)
Class for parsing fixed format input files.

Files

- file [nosh.h](#)
Contains declarations for class NOsh.
- file [nosh.c](#)
Class NOsh methods.

Defines

- #define [NOSH_MAXMOL](#) 20
Maximum number of molecules in a run.
- #define [NOSH_MAXCALC](#) 20

Maximum number of calculations in a run.

- #define **NOSH_MAXPRINT** 20

Maximum number of PRINT statements in a run.

- #define **NOSH_MAXPOP** 20

Maximum number of operations in a PRINT statement.

TypeDefs

- typedef enum **eNOsh_MolFormat** **NOsh_MolFormat**

Declare NOsh_MolFormat type.

- typedef enum **eNOsh_CalcType** **NOsh_CalcType**

Declare NOsh_CalcType type.

- typedef enum **eNOsh_ParmFormat** **NOsh_ParmFormat**

Declare NOsh_ParmFormat type.

- typedef enum **eNOsh_PrintType** **NOsh_PrintType**

Declare NOsh_PrintType type.

- typedef struct **sNOsh** **NOsh**

Declaration of the NOsh class as the NOsh structure.

- typedef struct **sNOsh_calc** **NOsh_calc**

Declaration of the NOsh_calc class as the NOsh_calc structure.

Enumerations

- enum **eNOsh_MolFormat** { **NMF_PQR** = 0, **NMF_PDB** = 1, **NMF_XML** = 2 }

Molecule file format types.

- enum **eNOsh_CalcType** { **NCT_MG** = 0, **NCT_FEM** = 1, **NCT_APOL** = 2 }

NOsh calculation types.

- enum **eNOsh_ParmFormat** { **NPF_FLAT** = 0, **NPF_XML** = 1 }

Parameter file format types.

- enum **eNOsh_PrintType** {

NPT_ENERGY = 0, **NPT_FORCE** = 1, **NPT_ELECENERGY**, **NPT_ELECFORCE**,
NPT_APOLENERGY, **NPT_APOLFORCE** }

NOsh print types.

Functions

- VEXTERNC char * NOsh_getMolpath (NOsh *thee, int imol)
Returns path to specified molecule.
- VEXTERNC char * NOsh_getDielXpath (NOsh *thee, int imap)
Returns path to specified x-shifted dielectric map.
- VEXTERNC char * NOsh_getDielYpath (NOsh *thee, int imap)
Returns path to specified y-shifted dielectric map.
- VEXTERNC char * NOsh_getDielZpath (NOsh *thee, int imap)
Returns path to specified z-shifted dielectric map.
- VEXTERNC char * NOsh_getKappapath (NOsh *thee, int imap)
Returns path to specified kappa map.
- VEXTERNC char * NOsh_getPotpath (NOsh *thee, int imap)
Returns path to specified potential map.
- VEXTERNC char * NOsh_getChargepath (NOsh *thee, int imap)
Returns path to specified charge distribution map.
- VEXTERNC NOsh_calc * NOsh_getCalc (NOsh *thee, int icalc)
Returns specified calculation object.
- VEXTERNC int NOsh_getDielfmt (NOsh *thee, int imap)
Returns format of specified dielectric map.
- VEXTERNC int NOsh_getKappafmt (NOsh *thee, int imap)
Returns format of specified kappa map.
- VEXTERNC int NOsh_getPotfmt (NOsh *thee, int imap)
Returns format of specified potential map.
- VEXTERNC int NOsh_getChargefmt (NOsh *thee, int imap)
Returns format of specified charge map.
- VEXTERNC NOsh_PrintType NOsh_printWhat (NOsh *thee, int iprint)
Return an integer ID of the observable to print .
- VEXTERNC char * NOsh_elecname (NOsh *thee, int ielec)
Return an integer mapping of an ELEC statement to a calculation ID .
- VEXTERNC int NOsh_elec2calc (NOsh *thee, int icalc)
Return the name of an elec statement.
- VEXTERNC int NOsh_apol2calc (NOsh *thee, int icalc)
Return the name of an apol statement.
- VEXTERNC int NOsh_printNarg (NOsh *thee, int iprint)
Return number of arguments to PRINT statement .
- VEXTERNC int NOsh_printOp (NOsh *thee, int iprint, int iarg)
Return integer ID for specified operation .
- VEXTERNC int NOsh_printCalc (NOsh *thee, int iprint, int iarg)
Return calculation ID for specified PRINT statement .

- VEXTERNC `NOsh * NOsh_ctor` (int rank, int size)
Construct NOsh.
- VEXTERNC `NOsh_calc * NOsh_calc_ctor (NOsh_CalcType calcType)`
Construct NOsh_calc.
- VEXTERNC int `NOsh_calc_copy (NOsh_calc *thee, NOsh_calc *source)`
Copy NOsh_calc object into thee.
- VEXTERNC void `NOsh_calc_dtor (NOsh_calc **thee)`
Object destructor.
- VEXTERNC int `NOsh_ctor2 (NOsh *thee, int rank, int size)`
FORTRAN stub to construct NOsh.
- VEXTERNC void `NOsh_dtor (NOsh **thee)`
Object destructor.
- VEXTERNC void `NOsh_dtor2 (NOsh *thee)`
FORTRAN stub for object destructor.
- VEXTERNC int `NOsh_parseInput (NOsh *thee, Vio *sock)`
Parse an input file from a socket.
- VEXTERNC int `NOsh_parseInputFile (NOsh *thee, char *filename)`
Parse an input file only from a file.
- VEXTERNC int `NOsh_setupElecCalc (NOsh *thee, Valist *alist[NOSH_MAXMOL])`
Setup the series of electrostatics calculations.
- VEXTERNC int `NOsh_setupApolCalc (NOsh *thee, Valist *alist[NOSH_MAXMOL])`
Setup the series of non-polar calculations.

8.7.1 Detailed Description

Class for parsing for fixed format input files.

8.7.2 Enumeration Type Documentation

8.7.2.1 enum eNOsh_CalcType

NOsh calculation types.

Enumerator:

- NCT_MG** Multigrid
- NCT_FEM** Finite element
- NCT_APOL** non-polar

Definition at line 112 of file `nosh.h`.

8.7.2.2 enum eNOsh_MolFormat

Molecule file format types.

Enumerator:

NMF_PQR PQR format

NMF_PDB PDB format

NMF_XML XML format

Definition at line 96 of file [nosh.h](#).

8.7.2.3 enum eNOsh_ParmFormat

Parameter file format types.

Enumerator:

NPF_FLAT Flat-file format

NPF_XML XML format

Definition at line 128 of file [nosh.h](#).

8.7.2.4 enum eNOsh_PrintType

NOsh print types.

Enumerator:

NPT_ENERGY Energy (deprecated)

NPT_FORCE Force (deprecated)

NPT_ELECENERGY Elec Energy

NPT_ELECFORCE Elec Force

NPT_APOLENERGY Apol Energy

NPT_APOLFORCE Apol Force

Definition at line 143 of file [nosh.h](#).

8.7.3 Function Documentation

8.7.3.1 VEXTERNC int NOsh_apol2calc (NOsh * *thee*, int *icalc*)

Return the name of an apol statement.

Author

David Gohara

Parameters

<i>thee</i>	NOsh object to use
<i>icalc</i>	ID of CALC statement

Returns

The name (if present) of an APOL statement

Definition at line 222 of file [nosh.c](#).

8.7.3.2 VEXTERNC int NOsh_calc_copy (NOsh_calc * *thee*, NOsh_calc * *source*)

Copy NOsh_calc object into thee.

Author

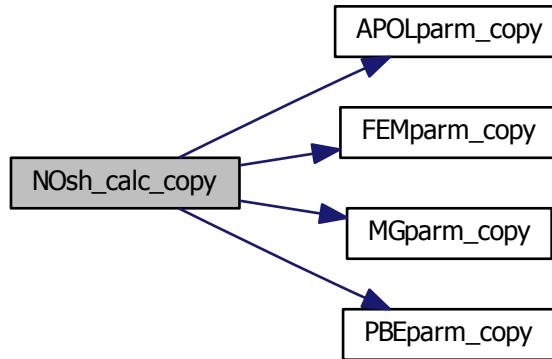
Nathan Baker

Parameters

<i>thee</i>	Target object
<i>source</i>	Source object

Definition at line 376 of file [nosh.c](#).

Here is the call graph for this function:



8.7.3.3 VEXTERNC NOsh_calc* NOsh_calc_ctor (NOsh_CalcType calcType)

Construct NOsh_calc.

Author

Nathan Baker

Parameters

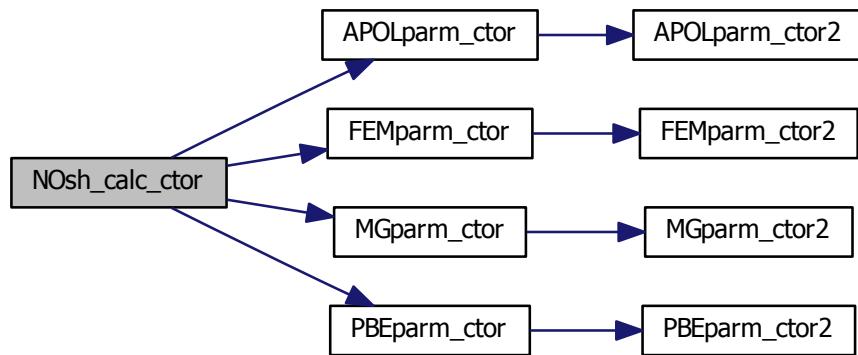
<i>calcType</i>	Calculation type
-----------------	------------------

Returns

Newly allocated and initialized NOsh object

Definition at line 314 of file [nosh.c](#).

Here is the call graph for this function:



8.7.3.4 VEXTERNC void NOsh_calc_dtor (NOsh_calc ** *thee*)

Object destructor.

Author

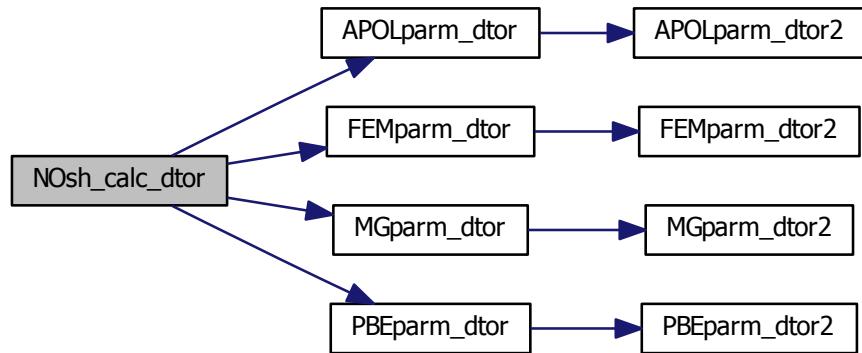
Nathan Baker

Parameters

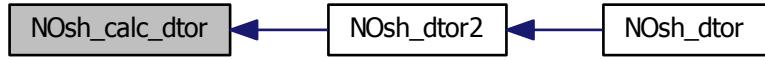
<i>thee</i>	Pointer to memory location of NOsh_calc object
-------------	--

Definition at line 346 of file [nosh.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.7.3.5 VEXTERNC NOsh* NOsh_ctor (int rank, int size)

Construct NOsh.

Author

Nathan Baker

Parameters

<i>rank</i>	Rank of current processor in parallel calculation (0 if not parallel)
<i>size</i>	Number of processors in parallel calculation (1 if not parallel)

Returns

Newly allocated and initialized NOsh object

Definition at line 248 of file [nosh.c](#).

Here is the call graph for this function:

**8.7.3.6 VEXTERNC int NOSh_ctor2 (NOSh * thee, int rank, int size)**

FORTRAN stub to construct NOsh.

Author

Nathan Baker

Parameters

<i>thee</i>	Space for NOsh objet
<i>rank</i>	Rank of current processor in parallel calculation (0 if not parallel)
<i>size</i>	Number of processors in parallel calculation (1 if not parallel)

Returns

1 if successful, 0 otherwise

Definition at line 259 of file [nosh.c](#).

Here is the caller graph for this function:



8.7.3.7 VEXTERNC void NOsh_dtor (NOsh ** thee)

Object destructor.

Author

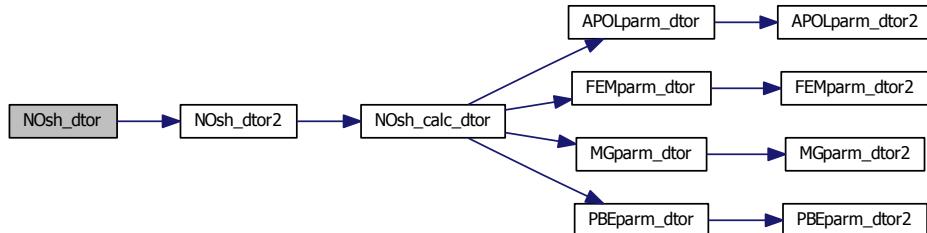
Nathan Baker

Parameters

thee	Pointer to memory location of NOsh object
------	---

Definition at line 294 of file [nosh.c](#).

Here is the call graph for this function:



8.7.3.8 VEXTERNC void NOsh_dtor2 (NOsh * *thee*)

FORTRAN stub for object destructor.

Author

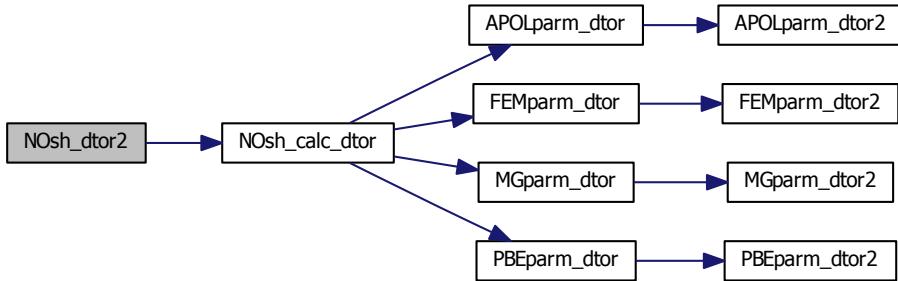
Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
-------------	------------------------

Definition at line [302](#) of file [nosh.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.7.3.9 VEXTERNC int NOsh_elec2calc (NOsh * *thee*, int *icalc*)

Return the name of an elec statement.

Author

Todd Dolinsky

Parameters

<i>thee</i>	NOsh object to use
<i>icalc</i>	ID of CALC statement

Returns

The name (if present) of an ELEC statement

Definition at line [216](#) of file [nosh.c](#).

8.7.3.10 VEXTERNC char* NOsh_elecname (NOsh * *thee*, int *ielec*)

Return an integer mapping of an ELEC statement to a calculation ID (.

See also

[elec2calc](#))

Author

Nathan Baker

Parameters

<i>thee</i>	NOsh object to use
<i>ielec</i>	ID of ELEC statement

Returns

An integer mapping of an ELEC statement to a calculation ID (

See also

[elec2calc](#))

Definition at line [228](#) of file [nosh.c](#).

8.7.3.11 VEXTERNC NOsh_calc* NOsh_getCalc (NOsh * *thee*, int *icalc*)

Returns specified calculation object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>icalc</i>	Calculation ID of interest

Returns

Pointer to specified calculation object

Definition at line 175 of file [nosh.c](#).

8.7.3.12 VEXTERNC int NOsh_getChargefmt (NOsh * *thee*, int *imap*)

Returns format of specified charge map.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Calculation ID of interest

Returns

Format of charge map

Definition at line 195 of file [nosh.c](#).

8.7.3.13 VEXTERNC char* NOsh_getChargepath (NOsh * *thee*, int *imap*)

Returns path to specified charge distribution map.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Map ID of interest

Returns

Path string

Definition at line [170](#) of file [nosh.c](#).

8.7.3.14 VEXTERNC int NOsh_getDielfmt (NOsh * *thee*, int *imap*)

Returns format of specified dielectric map.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Calculation ID of interest

Returns

Format of dielectric map

Definition at line [180](#) of file [nosh.c](#).

8.7.3.15 VEXTERNC char* NOsh_getDielXpath (NOsh * *thee*, int *imap*)

Returns path to specified x-shifted dielectric map.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Map ID of interest

Returns

Path string

Definition at line [145](#) of file [nosh.c](#).

8.7.3.16 VEXTERNC char* NOsh_getDieIYpath (NOsh * *thee*, int *imap*)

Returns path to specified y-shifted dielectric map.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Map ID of interest

Returns

Path string

Definition at line 150 of file [nosh.c](#).

8.7.3.17 VEXTERNC char* NOsh_getDieIZpath (NOsh * *thee*, int *imap*)

Returns path to specified z-shifted dielectric map.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Map ID of interest

Returns

Path string

Definition at line 155 of file [nosh.c](#).

8.7.3.18 VEXTERNC int NOsh_getKappafmt (NOsh * *thee*, int *imap*)

Returns format of specified kappa map.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Calculation ID of interest

Returns

Format of kappa map

Definition at line [185](#) of file [nosh.c](#).

8.7.3.19 VEXTERNC char* NOsh_getKappapath (NOsh * *thee*, int *imap*)

Returns path to specified kappa map.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Map ID of interest

Returns

Path string

Definition at line [160](#) of file [nosh.c](#).

8.7.3.20 VEXTERNC char* NOsh_getMolpath (NOsh * *thee*, int *imol*)

Returns path to specified molecule.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imol</i>	Molecule ID of interest

Returns

Path string

Definition at line [140](#) of file [nosh.c](#).

8.7.3.21 VEXTERNC int NOsh_getPotfmt (NOsh * *thee*, int *imap*)

Returns format of specified potential map.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Calculation ID of interest

Returns

Format of potential map

Definition at line 190 of file [nosh.c](#).

8.7.3.22 VEXTERNC char* NOsh_getPotpath (NOsh * *thee*, int *imap*)

Returns path to specified potential map.

Author

David Gohara

Parameters

<i>thee</i>	Pointer to NOsh object
<i>imap</i>	Map ID of interest

Returns

Path string

Definition at line 165 of file [nosh.c](#).

8.7.3.23 VEXTERNC int NOsh_parseInput (NOsh * *thee*, Vio * *sock*)

Parse an input file from a socket.

Note

Should be called before NOsh_setupCalc

Author

Nathan Baker and Todd Dolinsky

Parameters

<i>thee</i>	Pointer to NOsh object
<i>sock</i>	Stream of tokens to parse

Returns

1 if successful, 0 otherwise

Definition at line 412 of file [nosh.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.7.3.24 VEXTERNC int NOsh_parseInputFile (NOsh * *thee*, char * *filename*)

Parse an input file only from a file.

Note

Included for SWIG wrapper compatibility
Should be called before NOsh_setupCalc

Author

Nathan Baker and Todd Dolinsky

Parameters

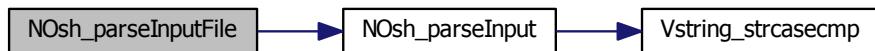
<i>thee</i>	Pointer to NOsh object
<i>filename</i>	Name/path of readable file

Returns

1 if successful, 0 otherwise

Definition at line 397 of file [nosh.c](#).

Here is the call graph for this function:



8.7.3.25 VEXTERNC int NOsh_printCalc(NOsh * *thee*, int *iprint*, int *iarg*)

Return calculation ID for specified PRINT statement (.

See also

[printcalc\(\)](#)

Author

Nathan Baker

Parameters

<i>thee</i>	NOsh object to use
<i>iprint</i>	ID of PRINT statement
<i>iarg</i>	ID of operation in PRINT statement

Returns

Calculation ID for specified PRINT statement (

See also

[printcalc\(\)](#)

Definition at line 241 of file [nosh.c](#).

8.7.3.26 VEXTERNC int NOsh_printNarg (NOsh * *thee*, int *iprint*)

Return number of arguments to PRINT statement (.

See also

[printnarg](#))

Author

Nathan Baker

Parameters

<i>thee</i>	NOsh object to use
<i>iprint</i>	ID of PRINT statement

Returns

Number of arguments to PRINT statement (

See also

[printnarg](#))

Definition at line 210 of file [nosh.c](#).

8.7.3.27 VEXTERNC int NOsh_printOp (NOsh * *thee*, int *iprint*, int *iarg*)

Return integer ID for specified operation (.

See also

[printop](#))

Author

Nathan Baker

Parameters

<i>thee</i>	NOsh object to use
<i>iprint</i>	ID of PRINT statement
<i>iarg</i>	ID of operation in PRINT statement

Returns

Integer ID for specified operation (

See also

printop)

Definition at line [234](#) of file [nosh.c](#).

8.7.3.28 VEXTERNC NOsh_PrintType NOsh_printWhat (NOsh * *thee*, int *iprint*)

Return an integer ID of the observable to print (.

See also

printwhat)

Author

Nathan Baker

Parameters

<i>thee</i>	NOsh object to use
<i>iprint</i>	ID of PRINT statement

Returns

An integer ID of the observable to print (

See also

printwhat)

Definition at line [204](#) of file [nosh.c](#).

8.7.3.29 VEXTERNC int NOsh_setupApolCalc (NOsh * *thee*, Valist * *alist*[NOSH_MAXMOL])

Setup the series of non-polar calculations.

Note

Should be called after NOsh_parseInput*

Author

Nathan Baker and Todd Dolinsky

Parameters

<i>thee</i>	Pointer to NOsh object
<i>alist</i>	Array of pointers to Valist objects (molecules used to center mesh);

Returns

1 if successful, 0 otherwise

Parameters

<i>thee</i>	NOsh object
<i>alist</i>	Atom list for calculation

Definition at line 1296 of file [nosh.c](#).

8.7.3.30 VEXTERNC int NOsh_setupElecCalc (NOsh * *thee*, Valist * *alist*[NOSH_MAXMOL])

Setup the series of electrostatics calculations.

Note

Should be called after NOsh_parseInput*

Author

Nathan Baker and Todd Dolinsky

Parameters

<i>thee</i>	Pointer to NOsh object
<i>alist</i>	Array of pointers to Valist objects (molecules used to center mesh);

Returns

1 if successful, 0 otherwise

Parameters

<i>thee</i>	NOsh object
<i>alist</i>	Atom list for calculation

Definition at line 1213 of file [nosh.c](#).

8.8 PBEparm class

Parameter structure for PBE variables independent of solver.

Data Structures

- struct `sPBEparm`

Parameter structure for PBE variables from input files.

Files

- file `pbeparm.h`

Contains declarations for class `PBEparm`.

- file `pbeparm.c`

Class `PBEparm` methods.

Defines

- `#define PBEPARM_MAXWRITE 20`

Number of things that can be written out in a single calculation.

Typedefs

- `typedef enum ePBEparm_calcEnergy PBEparm_calcEnergy`

Define `ePBEparm_calcEnergy` enumeration as `PBEparm_calcEnergy`.

- `typedef enum ePBEparm_calcForce PBEparm_calcForce`

Define `ePBEparm_calcForce` enumeration as `PBEparm_calcForce`.

- `typedef struct sPBEparm PBEparm`

Declaration of the `PBEparm` class as the `PBEparm` structure.

Enumerations

- `enum ePBEparm_calcEnergy { PCE_NO = 0, PCE_TOTAL = 1, PCE_COMPS = 2 }`

Define energy calculation enumeration.

- `enum ePBEparm_calcForce { PCF_NO = 0, PCF_TOTAL = 1, PCF_COMPS = 2 }`

Define force calculation enumeration.

Functions

- VEXTERNC double `PBEParm_getIonCharge` (`PBEParm *thee, int iion`)
Get charge (e) of specified ion species.
- VEXTERNC double `PBEParm_getIonConc` (`PBEParm *thee, int iion`)
Get concentration (M) of specified ion species.
- VEXTERNC double `PBEParm_getIonRadius` (`PBEParm *thee, int iion`)
Get radius (A) of specified ion species.
- VEXTERNC `PBEParm * PBEParm_ctor ()`
Construct PBEParm object.
- VEXTERNC int `PBEParm_ctor2` (`PBEParm *thee`)
FORTRAN stub to construct PBEParm object.
- VEXTERNC void `PBEParm_dtor` (`PBEParm **thee`)
Object destructor.
- VEXTERNC void `PBEParm_dtor2` (`PBEParm *thee`)
FORTRAN stub for object destructor.
- VEXTERNC int `PBEParm_check` (`PBEParm *thee`)
Consistency check for parameter values stored in object.
- VEXTERNC void `PBEParm_copy` (`PBEParm *thee, PBEParm *parm`)
Copy PBEParm object into thee.
- VEXTERNC int `PBEParm_parseToken` (`PBEParm *thee, char tok[VMAX_BUFSIZE], Vio *sock`)
Parse a keyword from an input file.

8.8.1 Detailed Description

Parameter structure for PBE variables independent of solver.

8.8.2 Enumeration Type Documentation

8.8.2.1 enum ePBEParm_calcEnergy

Define energy calculation enumeration.

Enumerator:

- PCE_NO** Do not perform energy calculation
- PCE_TOTAL** Calculate total energy only
- PCE_COMPS** Calculate per-atom energy components

Definition at line 80 of file `pbeparm.h`.

8.8.2.2 enum ePBEparm_calcForce

Define force calculation enumeration.

Enumerator:

PCF_NO Do not perform force calculation

PCF_TOTAL Calculate total force only

PCF_COMP Calculate per-atom force components

Definition at line 96 of file [pbeparm.h](#).

8.8.3 Function Documentation

8.8.3.1 VEXTERNC int PBEparm_check (PBEparm * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Returns

1 if OK, 0 otherwise

Parameters

<i>thee</i>	Object to be checked
-------------	----------------------

Definition at line 185 of file [pbeparm.c](#).

8.8.3.2 VEXTERNC void PBEparm_copy (PBEparm * *thee*, PBEparm * *parm*)

Copy PBEparm object into thee.

Author

Nathan Baker

Parameters

<i>thee</i>	Target for copy
<i>parm</i>	Source for copy

Definition at line 285 of file pbeparm.c.

Here is the caller graph for this function:



8.8.3.3 VEXTERNC PBEparm* PBEparm_ctor()

Construct PBEparm object.

Author

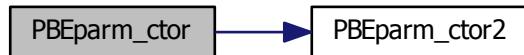
Nathan Baker

Returns

Newly allocated and initialized PBEparm object

Definition at line 106 of file pbeparm.c.

Here is the call graph for this function:



Here is the caller graph for this function:



8.8.3.4 VEXTERNC int PBEparm_ctor2 (PBEparm * *thee*)

FORTRAN stub to construct PBEparm object.

Author

Nathan Baker

Returns

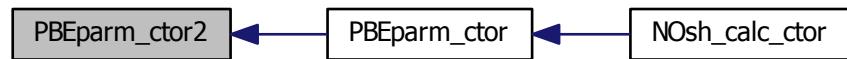
1 if successful, 0 otherwise

Parameters

<i>thee</i>	Memory location for object
-------------	----------------------------

Definition at line 117 of file [pbeparm.c](#).

Here is the caller graph for this function:



8.8.3.5 VEXTERNC void PBparm_dtor (PBparm ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

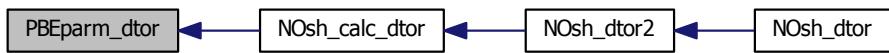
<i>thee</i>	Pointer to memory location of object
-------------	--------------------------------------

Definition at line 175 of file [pbparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**8.8.3.6 VEXTERNC void PBparm_dtor2 (PBparm * *thee*)**

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
-------------	-----------------------------------

Definition at line 183 of file [pbeparm.c](#).

Here is the caller graph for this function:



8.8.3.7 VEXTERNC double PBEParm_getIonCharge (PBEParm * *thee*, int *iion*)

Get charge (e) of specified ion species.

Author

Nathan Baker

Returns

Charge of ion species (e)

Parameters

<i>thee</i>	PBEParm object
<i>iion</i>	Ion species ID/index

Definition at line 67 of file [pbeparm.c](#).

8.8.3.8 VEXTERNC double PBEParm_getIonConc (PBEParm * *thee*, int *iion*)

Get concentration (M) of specified ion species.

Author

Nathan Baker

Returns

Concentration of ion species (M)

Parameters

<i>thee</i>	PBEparm object
<i>iion</i>	Ion species ID/index

Definition at line 73 of file [pbeparm.c](#).

8.8.3.9 VEXTERNC double PBEparm_getIonRadius (PBEparm * *thee*, int *iion*)

Get radius (A) of specified ion species.

Author

Nathan Baker

Returns

Radius of ion species (A)

Parameters

<i>thee</i>	PBEparm object
<i>iion</i>	Ion species ID/index

Definition at line 79 of file [pbeparm.c](#).

8.8.3.10 VEXTERNC int PBEparm_parseToken (PBEparm * *thee*, char *tok*[VMAX_BUFSIZE], Vio * *sock*)

Parse a keyword from an input file.

Author

Nathan Baker

Returns

1 if matched and assigned; -1 if matched, but there's some sort of error (i.e., too few args); 0 if not matched

Parameters

<i>thee</i>	Parsing object
<i>tok</i>	Token to parse
<i>sock</i>	Socket for additional tokens

Definition at line 1208 of file [pbeparm.c](#).

Here is the call graph for this function:



8.9 Vacc class

Solvent- and ion-accessibility oracle.

Data Structures

- struct [sVaccSurf](#)
Surface object list of per-atom surface points.
- struct [sVacc](#)
Oracle for solvent- and ion-accessibility around a biomolecule.

Files

- file [vacc.h](#)
Contains declarations for class Vacc.
- file [vacc.c](#)
Class Vacc methods.

Typedefs

- typedef struct [sVaccSurf](#) [VaccSurf](#)
Declaration of the VaccSurf class as the VaccSurf structure.
- typedef struct [sVacc](#) [Vacc](#)
Declaration of the Vacc class as the Vacc structure.

Functions

- VEXTERNC unsigned long int `Vacc_memChk (Vacc *thee)`
Get number of bytes in this object and its members.
- VEXTERNC `VaccSurf * VaccSurf_ctor (Vmem *mem, double probe_radius, int nsphere)`
Allocate and construct the surface object; do not assign surface points to positions.
- VEXTERNC int `VaccSurf_ctor2 (VaccSurf *thee, Vmem *mem, double probe_radius, int nsphere)`
Construct the surface object using previously allocated memory; do not assign surface points to positions.
- VEXTERNC void `VaccSurf_dtor (VaccSurf **thee)`
Destroy the surface object and free its memory.
- VEXTERNC void `VaccSurf_dtor2 (VaccSurf *thee)`
Destroy the surface object.
- VEXTERNC `VaccSurf * VaccSurf_refSphere (Vmem *mem, int npts)`
Set up an array of points for a reference sphere of unit radius.
- VEXTERNC `VaccSurf * Vacc_atomSurf (Vacc *thee, Vatom *atom, VaccSurf *ref, double probe_radius)`
Set up an array of points corresponding to the SAS due to a particular atom.
- VEXTERNC `Vacc * Vacc_ctor (Valist *alist, Vclist *clist, double surf_density)`
Construct the accessibility object.
- VEXTERNC int `Vacc_ctor2 (Vacc *thee, Valist *alist, Vclist *clist, double surf_density)`
FORTRAN stub to construct the accessibility object.
- VEXTERNC void `Vacc_dtor (Vacc **thee)`
Destroy object.
- VEXTERNC void `Vacc_dtor2 (Vacc *thee)`
FORTRAN stub to destroy object.
- VEXTERNC double `Vacc_vdwAcc (Vacc *thee, double center[VAPBS_DIM])`
Report van der Waals accessibility.
- VEXTERNC double `Vacc_ivdwAcc (Vacc *thee, double center[VAPBS_DIM], double radius)`
Report inflated van der Waals accessibility.
- VEXTERNC double `Vacc_molAcc (Vacc *thee, double center[VAPBS_DIM], double radius)`
Report molecular accessibility.
- VEXTERNC double `Vacc_fastMolAcc (Vacc *thee, double center[VAPBS_DIM], double radius)`
Report molecular accessibility quickly.

- VEXTERNC double `Vacc_splineAcc` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad)`
Report spline-based accessibility.
- VEXTERNC void `Vacc_splineAccGrad` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, double *grad)`
Report gradient of spline-based accessibility.
- VEXTERNC double `Vacc_splineAccAtom` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom)`
Report spline-based accessibility for a given atom.
- VEXTERNC void `Vacc_splineAccGradAtomUnnorm` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom, double *force)
Report gradient of spline-based accessibility with respect to a particular atom (see Vpmg_splineAccAtom)`
- VEXTERNC void `Vacc_splineAccGradAtomNorm` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom, double *force)
Report gradient of spline-based accessibility with respect to a particular atom normalized by the accessibility value due to that atom at that point (see Vpmg_splineAccAtom)`
- VEXTERNC void `Vacc_splineAccGradAtomNorm4` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom, double *force)
Report gradient of spline-based accessibility with respect to a particular atom normalized by a 4th order accessibility value due to that atom at that point (see Vpmg_splineAccAtom)`
- VEXTERNC void `Vacc_splineAccGradAtomNorm3` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom, double *force)
Report gradient of spline-based accessibility with respect to a particular atom normalized by a 3rd order accessibility value due to that atom at that point (see Vpmg_splineAccAtom)`
- VEXTERNC double `Vacc_SASA` (`Vacc *thee, double radius)`
Build the solvent accessible surface (SAS) and calculate the solvent accessible surface area.
- VEXTERNC double `Vacc_totalSASA` (`Vacc *thee, double radius)`
Return the total solvent accessible surface area (SASA)
- VEXTERNC double `Vacc_atomSASA` (`Vacc *thee, double radius, Vatom *atom)
Return the atomic solvent accessible surface area (SASA)`
- VEXTERNC `VaccSurf * Vacc_atomSASPoints` (`Vacc *thee, double radius, Vatom *atom)
Get the set of points for this atom's solvent-accessible surface.`
- VEXTERNC void `Vacc_atomdSAV` (`Vacc *thee, double radius, Vatom *atom, double *dSA)
Get the derivative of solvent accessible volume.`
- VEXTERNC void `Vacc_atomdSASA` (`Vacc *thee, double dpos, double radius, Vatom *atom, double *dSA)`

Get the derivative of solvent accessible area.

- VEXTERNC void `Vacc_totalAtomdSASA` (`Vacc *thee`, double `dpos`, double `radius`, `Vatom *atom`, double `*dSA`)

Testing purposes only.

- VEXTERNC void `Vacc_totalAtomdSAV` (`Vacc *thee`, double `dpos`, double `radius`, `Vatom *atom`, double `*dSA`, `Vclist *clist`)

Total solvent accessible volume.

- VEXTERNC double `Vacc_totalSAV` (`Vacc *thee`, `Vclist *clist`, `APOLparm *apolparm`, double `radius`)

Return the total solvent accessible volume (SAV)

- VEXTERNC int `Vacc_wcaEnergy` (`Vacc *thee`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`)

Return the WCA integral energy.

- VEXTERNC int `Vacc_wcaForceAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Vclist *clist`, `Vatom *atom`, double `*force`)

Return the WCA integral force.

- VEXTERNC int `Vacc_wcaEnergyAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`, int `iatom`, double `*value`)

Calculate the WCA energy for an atom.

8.9.1 Detailed Description

Solvent- and ion-accessibility oracle.

8.9.2 Function Documentation

8.9.2.1 VEXTERNC void `Vacc_atomdSASA` (`Vacc *thee`, double `dpos`, double `radius`, `Vatom *atom`, double `*dSA`)

Get the derivative of solvent accessible area.

Author

Jason Wagoner, David Gohara, Nathan Baker

Parameters

<code>thee</code>	Acessibility object
<code>dpos</code>	Atom position offset
<code>radius</code>	Probe radius (Å)
<code>atom</code>	Atom of interest
<code>dSA</code>	Array holding answers of calc

Definition at line 1260 of file [vacc.c](#).

8.9.2.2 VEXTERNC void Vacc_atomdSAV (Vacc * *thee*, double *radius*, Vatom * *atom*, double * *dSA*)

Get the derivative of solvent accessible volume.

Author

Jason Wagoner, Nathan Baker

Parameters

<i>thee</i>	Acessibility object
<i>radius</i>	Probe radius (\AA)
<i>atom</i>	Atom of interest
<i>dSA</i>	Array holding answers of calc

Definition at line 1149 of file [vacc.c](#).

8.9.2.3 VEXTERNC double Vacc_atomSASA (Vacc * *thee*, double *radius*, Vatom * *atom*)

Return the atomic solvent accessible surface area (SASA)

Note

Alias for Vacc_SASA

Author

Nathan Baker

Returns

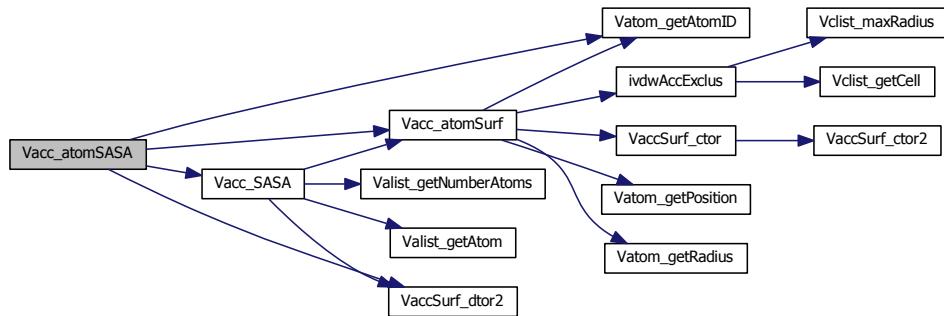
Atomic solvent accessible area (\AA^2)

Parameters

<i>thee</i>	Accessibility object
<i>radius</i>	Probe molecule radius (\AA)
<i>atom</i>	Atom of interest

Definition at line 716 of file [vacc.c](#).

Here is the call graph for this function:



8.9.2.4 VEXTERNC VaccSurf* Vacc_atomSASPoints (Vacc * *thee*, double *radius*, Vatom * *atom*)

Get the set of points for this atom's solvent-accessible surface.

Author

Nathan Baker

Returns

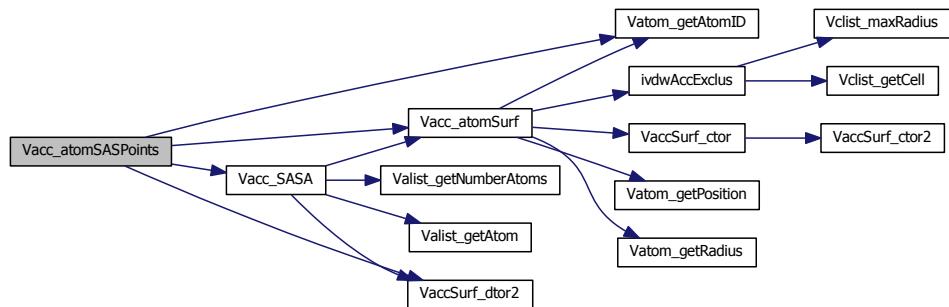
Pointer to VaccSurf object for this atom

Parameters

<i>thee</i>	Accessibility object
<i>radius</i>	Probe molecule radius (Å)
<i>atom</i>	Atom of interest

Definition at line 931 of file [vacc.c](#).

Here is the call graph for this function:



8.9.2.5 VEXTERNC VaccSurf* Vacc_atomSurf (Vacc * *thee*, Vatom * *atom*, VaccSurf * *ref*, double *probe_radius*)

Set up an array of points corresponding to the SAS due to a particular atom.

Author

Nathan Baker

Returns

Atom sphere surface object

Parameters

<i>thee</i>	Accessibility object for molecule
<i>atom</i>	Atom for which the surface should be constructed
<i>ref</i>	Reference sphere which sets the resolution for the surface.

See also

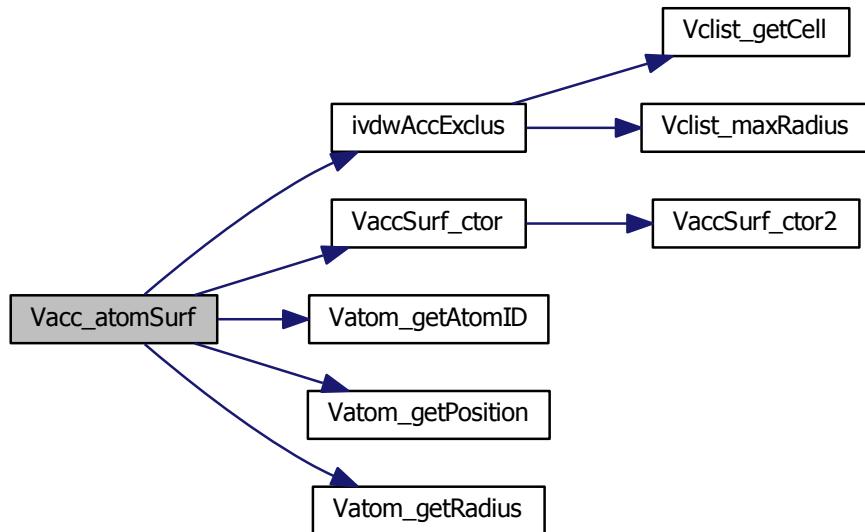
VaccSurf_refSphere

Parameters

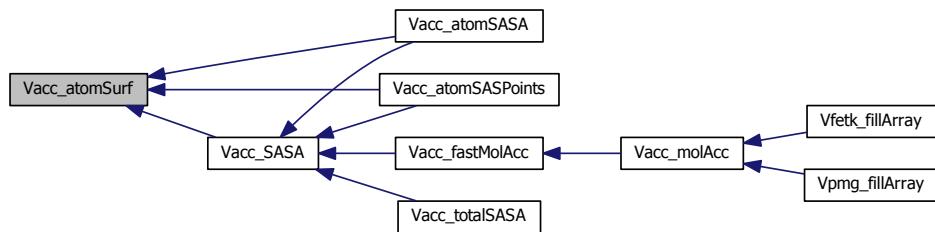
probe_radius Probe radius (in Å)

Definition at line 819 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.6 VEXTERNC Vacc* Vacc_ctor (Valist * *alist*, Vclist * *clist*, double *surf_density*)

Construct the accessibility object.

Author

Nathan Baker

Returns

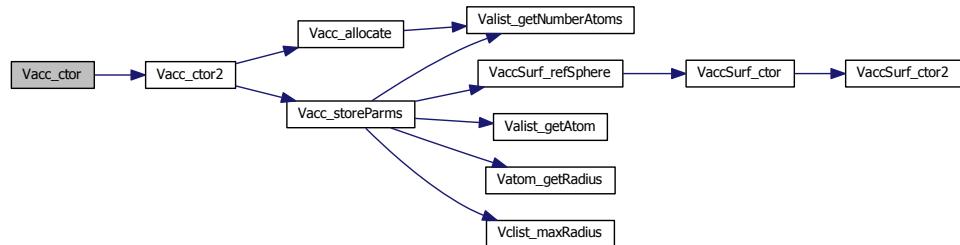
Newly allocated Vacc object

Parameters

<i>alist</i>	Molecule for accessibility queries
<i>clist</i>	Pre-constructed cell list for looking up atoms near specific positions
<i>surf_density</i>	Minimum per-atom solvent accessible surface point density (in pts/A ²)

Definition at line 132 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
8.9.2.7 VEXTERNC int Vacc_ctor2 ( Vacc * thee, Valist * alist, Vclist * clist, double
surf_density )
```

FORTRAN stub to construct the accessibility object.

Author

Nathan Baker

Returns

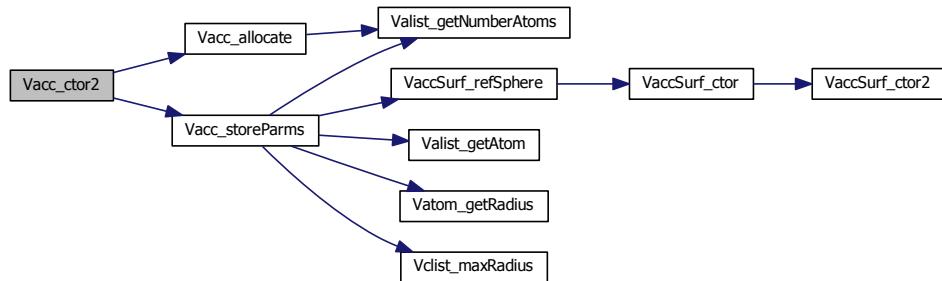
1 if successful, 0 otherwise

Parameters

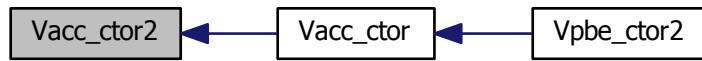
<i>thee</i>	Memory for Vacc objet
<i>alist</i>	Molecule for accessibility queries
<i>clist</i>	Pre-constructed cell list for looking up atoms near specific positions
<i>surf_density</i>	Minimum per-atom solvent accessible surface point density (in pts/A ²)

Definition at line 203 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.8 VEXTERNC void Vacc_dtor (Vacc ***thee*)

Destroy object.

Author

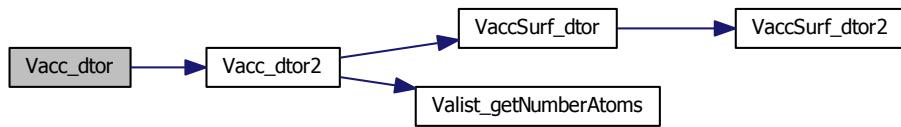
Nathan Baker

Parameters

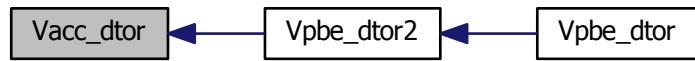
<i>thee</i>	Pointer to memory location of object
-------------	--------------------------------------

Definition at line 232 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.9 VEXTERNC void Vacc_dtor2 (Vacc * *thee*)

FORTRAN stub to destroy object.

Author

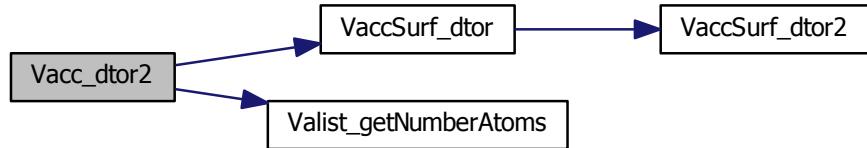
Nathan Baker

Parameters

<i>thee</i>	Pointer to object
-------------	-------------------

Definition at line 242 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.10 VEXTERNC double Vacc_fastMolAcc (*Vacc * thee*, double *center[VAPBS_DIM]*, double *radius*)

Report molecular accessibility quickly.

Given a point which is INSIDE the collection of inflated van der Waals spheres, but OUTSIDE the collection of non-inflated van der Waals spheres, determine accessibility of a probe (of radius *radius*) at a given point, given a collection of atomic spheres. Uses molecular (Connolly) surface definition.

Note

THIS ASSUMES YOU HAVE TESTED THAT THIS POINT IS DEFINITELY INSIDE THE INFLATED AND NON-INFLATED VAN DER WAALS SURFACES!

Author

Nathan Baker

Returns

Characteristic function value between 1.0 (accessible) and 0.0 (inaccessible)

Bug

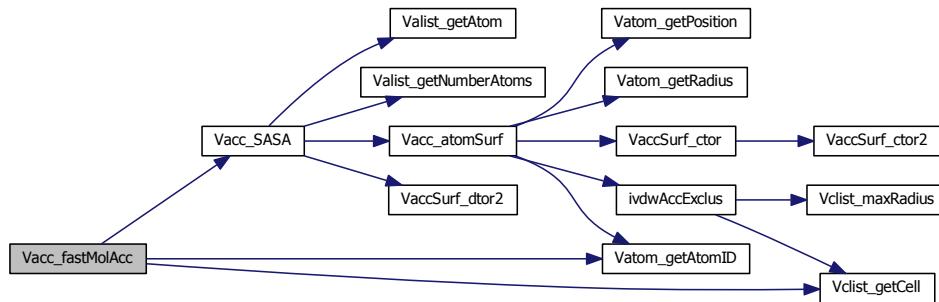
This routine has a slight bug which can generate very small internal regions of high dielectric (thanks to John Mongan and Jess Swanson for finding this)

Parameters

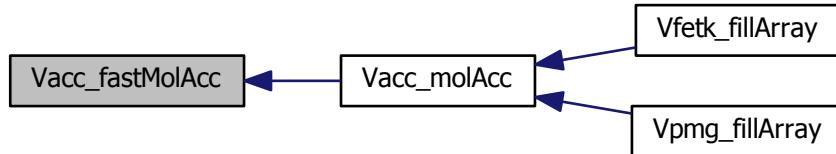
<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates
<i>radius</i>	Probe radius (in Å)

Definition at line 581 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.11 VEXTERNC double Vacc_ivdwAcc (Vacc * thee, double center[VAPBS_DIM], double radius)

Report inflated van der Waals accessibility.

Determines if a point is within the union of the spheres centered at the atomic centers with radii equal to the sum of the atomic van der Waals radius and the probe radius.

Author

Nathan Baker

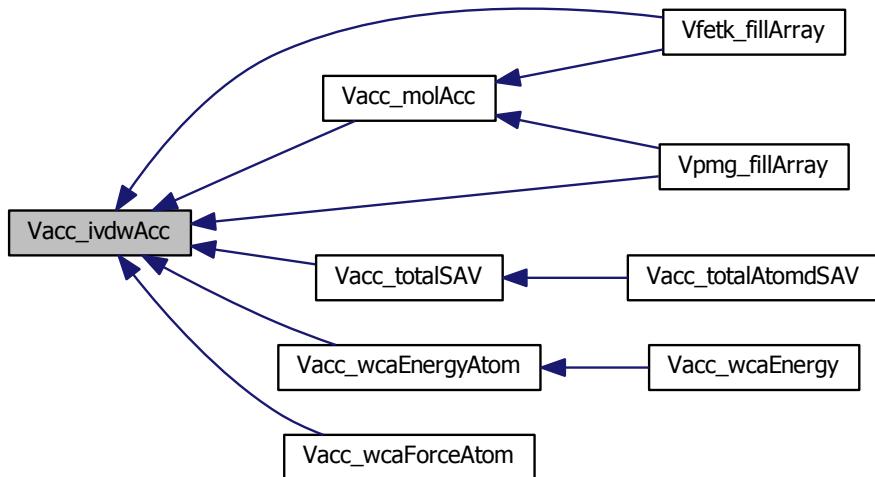
Returns

Characteristic function value between 1.0 (accessible) and 0.0 (inaccessible)

Parameters

<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates
<i>radius</i>	Probe radius (\AA)

Here is the caller graph for this function:



8.9.2.12 VEXTERNC unsigned long int Vacc_memChk (*Vacc* * *thee*)

Get number of bytes in this object and its members.

Author

Nathan Baker

Returns

Number of bytes allocated for object

Parameters

<i>thee</i>	Object for memory check
-------------	-------------------------

Definition at line 69 of file [vacc.c](#).

Here is the caller graph for this function:



8.9.2.13 VEXTERNC double Vacc_molAcc (**Vacc** * *thee*, double *center*[VAPBS_DIM], double *radius*)

Report molecular accessibility.

Determine accessibility of a probe (of radius *radius*) at a given point, given a collection of atomic spheres. Uses molecular (Connolly) surface definition.

Author

Nathan Baker

Returns

Characteristic function value between 1.0 (accessible) and 0.0 (inaccessible)

Bug

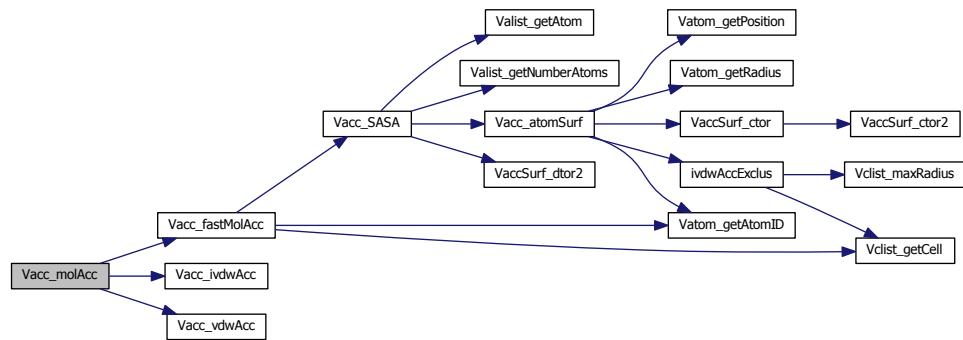
This routine has a slight bug which can generate very small internal regions of high dielectric (thanks to John Mongan and Jess Swanson for finding this)

Parameters

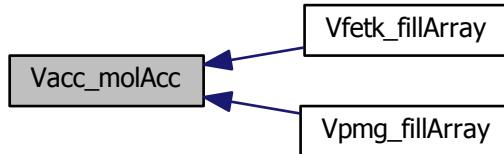
<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates
<i>radius</i>	Probe radius (in Å)

Definition at line 552 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.14 VEXTERNC double Vacc_SASA (*Vacc * thee, double radius*)

Build the solvent accessible surface (SAS) and calculate the solvent accessible surface area.

Note

Similar to UHBD FORTRAN routine by Brock Luty (returns UHBD's asas2)

Author

Nathan Baker (original FORTRAN routine by Brock Luty)

Returns

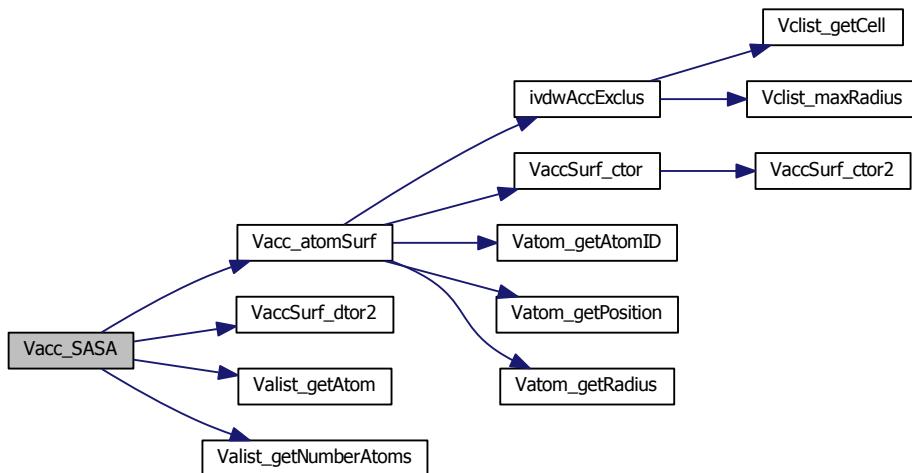
Total solvent accessible area (\AA^2)

Parameters

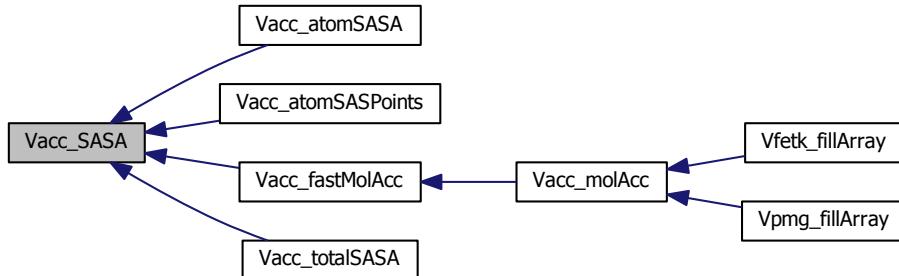
<i>thee</i>	Accessibility object
<i>radius</i>	Probe molecule radius (\AA)

Definition at line [657](#) of file `vacc.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.15 `VEXTERNC double Vacc_splineAcc (Vacc *thee, double center[VAPBS_DIM], double win, double inflrad)`

Report spline-based accessibility.

Determine accessibility at a given point, given a collection of atomic spheres. Uses Benoit Roux (Im et al, Comp Phys Comm, 111, 59–75, 1998) definition suitable for force evalution; basically a cubic spline.

Author

Nathan Baker

Returns

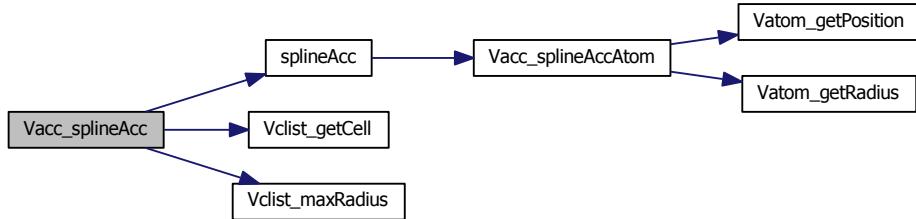
Characteristic function value between 1.0 (accessible) and 0.0 (inaccessible)

Parameters

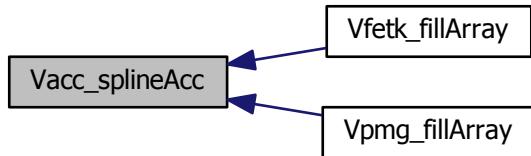
<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates
<i>win</i>	Spline window (Å)
<i>infrad</i>	Inflation radius (Å) for ion access.

Definition at line 472 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



**8.9.2.16 VEXTERNC double Vacc_splineAccAtom (Vacc * *thee*, double *center*[VAPBS_DIM],
double *win*, double *infrad*, Vatom * *atom*)**

Report spline-based accessibility for a given atom.

Determine accessibility at a given point for a given atomic spheres. Uses Benoit Roux (Im et al, Comp Phys Comm, 111, 59--75, 1998) definition suitable for force evalation; basically a cubic spline.

Author

Nathan Baker

Returns

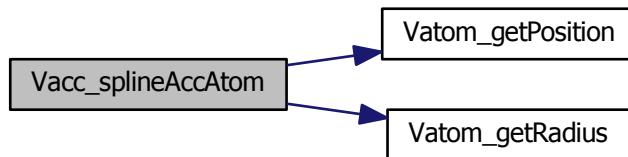
Characteristic function value between 1.0 (accessible) and 0.0 (inaccessible)

Parameters

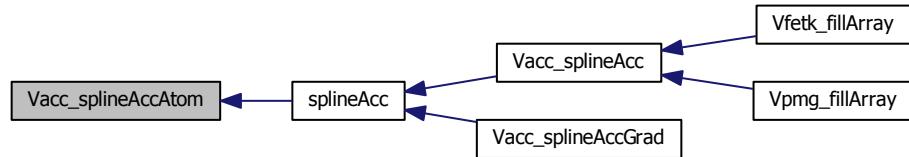
<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates
<i>win</i>	Spline window (\AA)
<i>infrad</i>	Inflation radius (\AA) for ion access.
<i>atom</i>	Atom

Definition at line 395 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
8.9.2.17 VEXTERNC void Vacc_splineAccGrad ( Vacc * thee, double center[VAPBS_DIM],
double win, double inflrad, double * grad )
```

Report gradient of spline-based accessibility.

Author

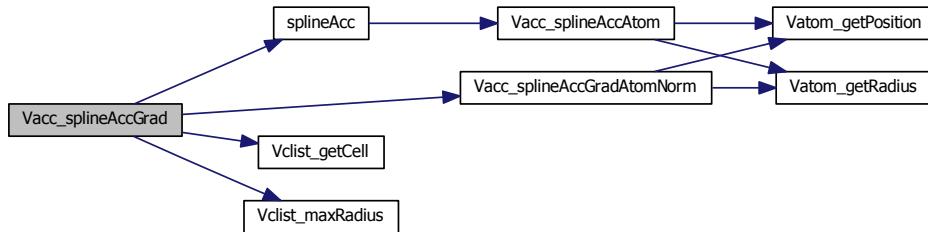
Nathan Baker

Parameters

<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates
<i>win</i>	Spline window (Å)
<i>infrad</i>	Inflation radius (Å) for ion access.
<i>grad</i>	3-vector set to gradient of accessibility

Definition at line 505 of file [vacc.c](#).

Here is the call graph for this function:



```
8.9.2.18 VEXTERNC void Vacc_splineAccGradAtomNorm ( Vacc * thee, double
center[VAPBS_DIM], double win, double inflrad, Vatom * atom, double * force )
```

Report gradient of spline-based accessibility with respect to a particular atom normalized by the accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)

Determine accessibility at a given point, given a collection of atomic spheres. Uses Benoit Roux (Im et al, Comp Phys Comm, 111, 59--75, 1998) definition suitable for force evalation; basically a cubic spline.

Author

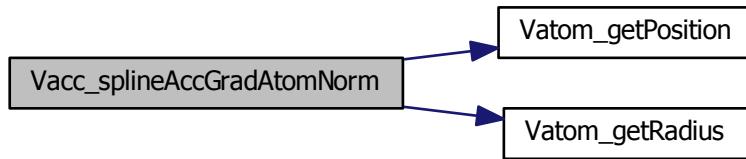
Nathan Baker

Parameters

<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates
<i>win</i>	Spline window (\AA)
<i>infrad</i>	Inflation radius (\AA) for ion access.
<i>atom</i>	Atom
<i>force</i>	VAPBS_DIM-vector set to gradient of accessibility

Definition at line 297 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
8.9.2.19 VEXTERNC void Vacc_splineAccGradAtomNorm3 ( Vacc * thee, double
center[VAPBS_DIM], double win, double infrad, Vatom * atom, double * force )
```

Report gradient of spline-based accessibility with respect to a particular atom normalized by a 3rd order accessibility value due to that atom at that point (see Vpmg_splineAccAtom)

Author

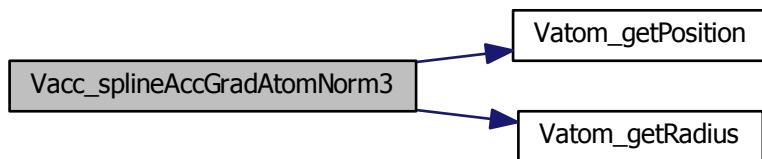
Michael Schnieders

Parameters

<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates
<i>win</i>	Spline window (Å)
<i>infrad</i>	Inflation radius (Å) for ion access.
<i>atom</i>	Atom
<i>force</i>	VAPBS_DIM-vector set to gradient of accessibility

Definition at line 1048 of file [vacc.c](#).

Here is the call graph for this function:



```
8.9.2.20 VEXTERNC void Vacc_splineAccGradAtomNorm4 ( Vacc * thee, double
center[VAPBS_DIM], double win, double infrad, Vatom * atom, double * force )
```

Report gradient of spline-based accessibility with respect to a particular atom normalized by a 4th order accessibility value due to that atom at that point (see Vpmg_splineAccAtom)

Author

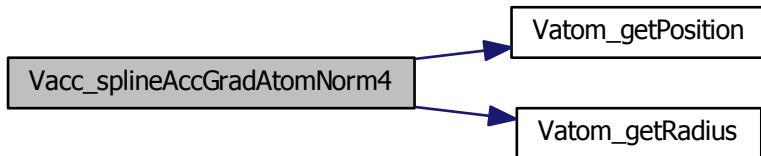
Michael Schnieders

Parameters

<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates
<i>win</i>	Spline window (\AA)
<i>infrad</i>	Inflation radius (\AA) for ion access.
<i>atom</i>	Atom
<i>force</i>	VAPBS_DIM-vector set to gradient of accessibility

Definition at line 955 of file [vacc.c](#).

Here is the call graph for this function:



8.9.2.21 VEXTERNC void Vacc_splineAccGradAtomUnnorm (**Vacc** * *thee*, double *center*[VAPBS_DIM], double *win*, double *infrad*, **Vatom** * *atom*, double * *force*)

Report gradient of spline-based accessibility with respect to a particular atom (see [Vpmg_splineAccAtom](#))

Determine accessibility at a given point, given a collection of atomic spheres. Uses Benoit Roux (Im et al, Comp Phys Comm, 111, 59–75, 1998) definition suitable for force evalution; basically a cubic spline.

Author

Nathan Baker

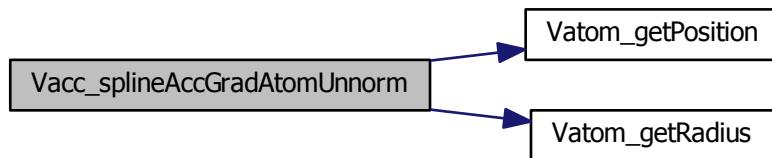
Parameters

<i>thee</i>	Accessibility object
-------------	----------------------

<i>center</i>	Probe center coordinates
<i>win</i>	Spline window (\AA)
<i>infrad</i>	Inflation radius (\AA) for ion access.
<i>atom</i>	Atom
<i>force</i>	VAPBS_DIM-vector set to gradient of accessibility

Definition at line 346 of file [vacc.c](#).

Here is the call graph for this function:



8.9.2.22 VEXTERNC void Vacc_totalAtomdSASA (*Vacc * thee, double dpos, double radius,*
*Vatom * atom, double * dSA*)

Testing purposes only.

Author

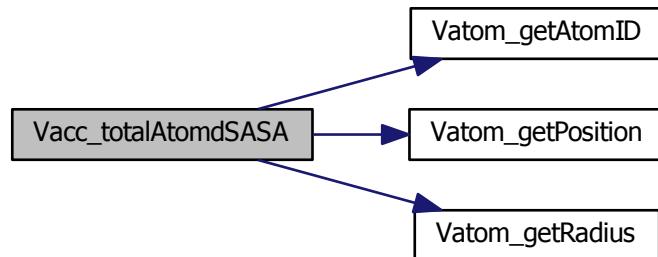
David Gohara, Nathan Baker

Parameters

<i>thee</i>	Acessibility object
<i>dpos</i>	Atom position offset
<i>radius</i>	Probe radius (\AA)
<i>atom</i>	Atom of interest
<i>dSA</i>	Array holding answers of calc

Definition at line 1322 of file [vacc.c](#).

Here is the call graph for this function:



8.9.2.23 `VEXTERNC void Vacc_totalAtomdSAV (Vacc * thee, double dpos, double radius,`
`Vatom * atom, double * dSA, Vclist * clist)`

Total solvent accessible volume.

Author

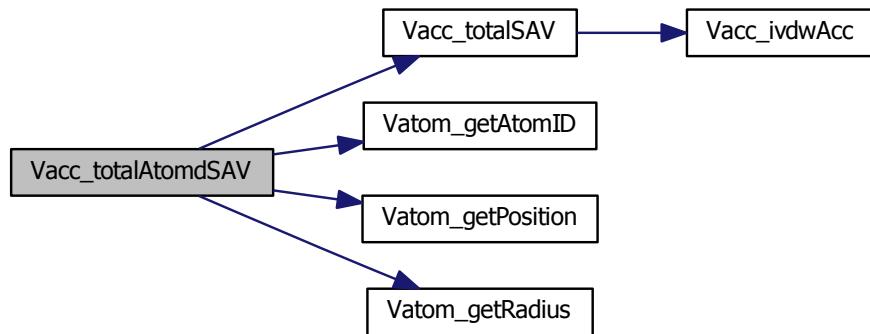
David Gohara, Nathan Baker

Parameters

<code><i>thee</i></code>	Acessibility object
<code><i>dpos</i></code>	Atom position offset
<code><i>radius</i></code>	Probe radius (Å)
<code><i>atom</i></code>	Atom of interest
<code><i>dSA</i></code>	Array holding answers of calc
<code><i>clist</i></code>	clist for this calculation

Definition at line 1381 of file [vacc.c](#).

Here is the call graph for this function:



8.9.2.24 VEXTERNC double Vacc_totalSASA (*Vacc * thee, double radius*)

Return the total solvent accessible surface area (SASA)

Note

Alias for Vacc_SASA

Author

Nathan Baker

Returns

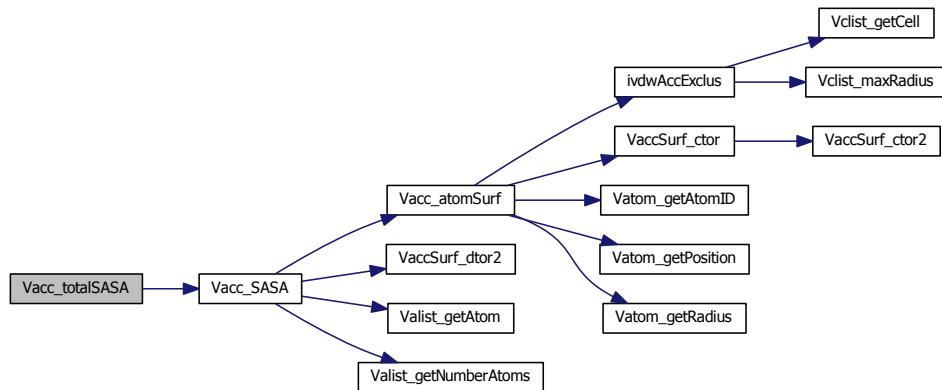
Total solvent accessible area (A^2)

Parameters

<i>thee</i>	Accessibility object
<i>radius</i>	Probe molecule radius (\AA)

Definition at line 710 of file [vacc.c](#).

Here is the call graph for this function:



8.9.2.25 VEXTERNC double Vacc_totalSAV (**Vacc** * *thee*, **Vclist** * *clist*, **APOLparm** * *apolparm*, double *radius*)

Return the total solvent accessible volume (SAV)

Note

Alias for Vacc_SAV

Author

David Gohara

Returns

Total solvent accessible volume (\AA^3)

Parameters

<i>thee</i>	Accessibility object
<i>clist</i>	Clist for acc object
<i>apolparm</i>	Apolar parameters -- could be VNULL if none required for this calculation. If VNULL, then default settings are used
<i>radius</i>	Probe molecule radius (\AA)

Definition at line 1436 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.26 VEXTERNC double Vacc_vdwAcc (Vacc * *thee*, double *center*[VAPBS_DIM])

Report van der Waals accessibility.

Determines if a point is within the union of the atomic spheres (with radii equal to their van der Waals radii).

Author

Nathan Baker

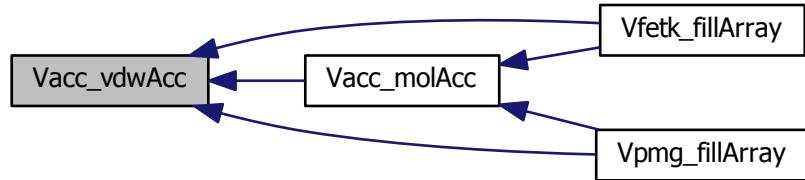
Returns

Characteristic function value between 1.0 (accessible) and 0.0 (inaccessible)

Parameters

<i>thee</i>	Accessibility object
<i>center</i>	Probe center coordinates

Here is the caller graph for this function:



8.9.2.27 `VEXTERNC int Vacc_wcaEnergy (Vacc * thee, APOLparm * apolparm, Valist * alist, Vclist * clist)`

Return the WCA integral energy.

Author

David Gohara

Returns

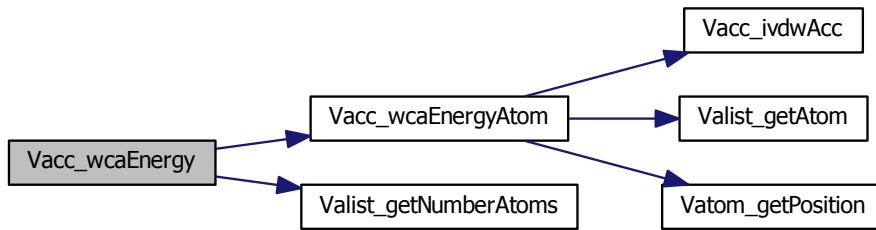
Success flag

Parameters

<i>thee</i>	Accessibility object
<i>apolparm</i>	Apolar calculation parameters
<i>alist</i>	Alist for acc object
<i>clist</i>	Clist for acc object

Definition at line 1654 of file `vacc.c`.

Here is the call graph for this function:



8.9.2.28 VEXTERNC int Vacc_wcaEnergyAtom (*Vacc * thee, APOLparm * apolparm, Valist * alist, Vclist * clist, int iatom, double * value*)

Calculate the WCA energy for an atom.

Author

Dave Gohara and Nathan Baker

Returns

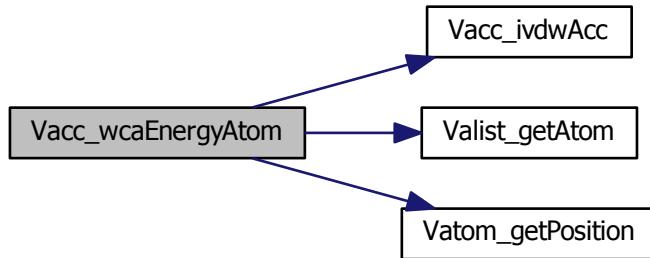
Success flag

Parameters

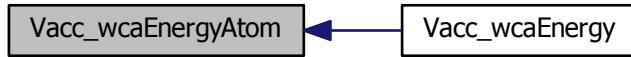
<i>thee</i>	Accessibility object
<i>apolparm</i>	Apolar calculation parameters
<i>alist</i>	Atom list
<i>clist</i>	Cell list associated with Vacc object
<i>iatom</i>	Index for atom of interest
<i>value</i>	Set to energy value

Definition at line 1513 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.29 VEXTERNC int Vacc_wcaForceAtom (*Vacc * thee, APOLparm * apolparm, Vclist * clist, Vatom * atom, double * force*)

Return the WCA integral force.

Author

David Gohara

Returns

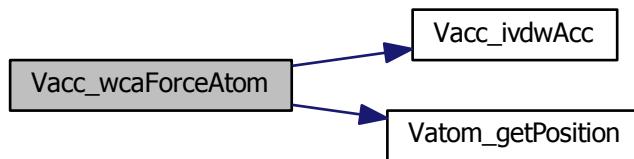
WCA energy (kJ/mol/A)

Parameters

<i>thee</i>	Accessibility object
<i>apolparm</i>	Apolar calculation parameters
<i>clist</i>	Clist for acc object
<i>atom</i>	Current atom
<i>force</i>	Force for atom

Definition at line 1689 of file [vacc.c](#).

Here is the call graph for this function:



8.9.2.30 VEXTERNC VaccSurf* VaccSurf_ctor (Vmem * mem, double probe_radius, int nsphere)

Allocate and construct the surface object; do not assign surface points to positions.

Author

Nathan Baker

Returns

Newly allocated and constructed surface object

Parameters

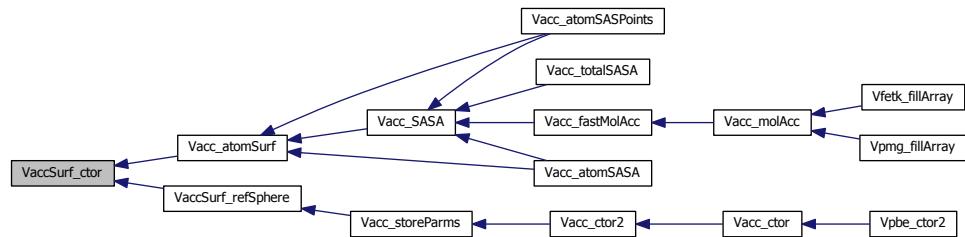
<i>mem</i>	Memory manager (can be VNULL)
<i>probe_-radius</i>	Probe radius (in A) for this surface
<i>nsphere</i>	Number of points in sphere

Definition at line 739 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.31 VEXTERNC int VaccSurf_ctor2 (VaccSurf * *thee*, Vmem * *mem*, double *probe_radius*, int *nsphere*)

Construct the surface object using previously allocated memory; do not assign surface points to positions.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

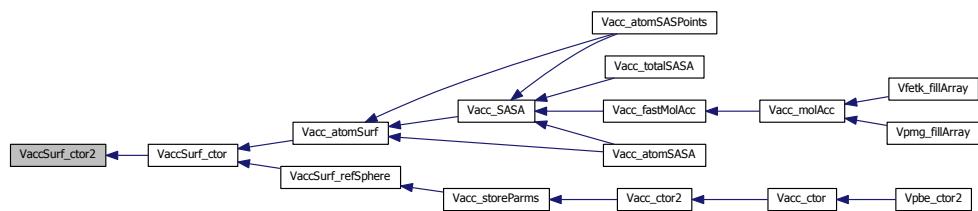
Parameters

<i>thee</i>	Allocated memory
-------------	------------------

<i>mem</i>	Memory manager (can be VNULL)
<i>probe_- radius</i>	Probe radius (in Å) for this surface
<i>nsphere</i>	Number of points in sphere

Definition at line 754 of file [vacc.c](#).

Here is the caller graph for this function:



8.9.2.32 VEXTERNC void VaccSurf_dtor (VaccSurf ** *thee*)

Destroy the surface object and free its memory.

Author

Nathan Baker

Parameters

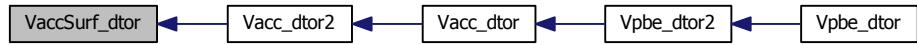
<i>thee</i>	Object to be destroyed
-------------	------------------------

Definition at line 785 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.33 VEXTERNC void VaccSurf_dtor2 (VaccSurf * *thee*)

Destroy the surface object.

Author

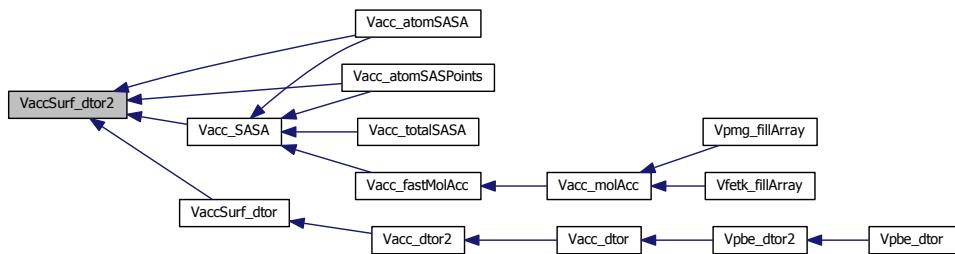
Nathan Baker

Parameters

<i>thee</i>	Object to be destroyed
-------------	------------------------

Definition at line 799 of file [vacc.c](#).

Here is the caller graph for this function:



8.9.2.34 VEXTERNC VaccSurf* VaccSurf_refSphere (Vmem * mem, int npts)

Set up an array of points for a reference sphere of unit radius.

Generates approximately npts # of points (actual number stored in thee->npts) somewhat uniformly distributed across a sphere of unit radius centered at the origin.

Note

This routine was shamelessly ripped off from sphere.f from UHBD as developed by Michael K. Gilson.

Author

Nathan Baker (original FORTRAN code by Mike Gilson)

Returns

Reference sphere surface object

Parameters

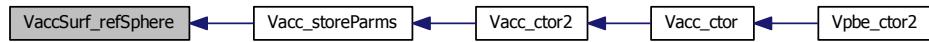
<i>mem</i>	Memory object
<i>npts</i>	Requested number of points on sphere

Definition at line 875 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.10 Valist class

Container class for list of atom objects.

Data Structures

- struct **sValist**

Container class for list of atom objects.

Files

- file **valist.h**

Contains declarations for class Valist.

TypeDefs

- typedef struct **sValist Valist**

Declaration of the Valist class as the Valist structure.

Functions

- VEXTERNC `Vatom * Valist_getAtomList (Valist *thee)`
Get actual array of atom objects from the list.
- VEXTERNC double `Valist_getCenterX (Valist *thee)`
Get x-coordinate of molecule center.
- VEXTERNC double `Valist_getCenterY (Valist *thee)`
Get y-coordinate of molecule center.
- VEXTERNC double `Valist_getCenterZ (Valist *thee)`
Get z-coordinate of molecule center.
- VEXTERNC int `Valist_getNumberAtoms (Valist *thee)`
Get number of atoms in the list.
- VEXTERNC `Vatom * Valist_getAtom (Valist *thee, int i)`
Get pointer to particular atom in list.
- VEXTERNC unsigned long int `Valist_memChk (Valist *thee)`
Get total memory allocated for this object and its members.
- VEXTERNC `Valist * Valist_ctor ()`
Construct the atom list object.
- VEXTERNC Vrc_Codes `Valist_ctor2 (Valist *thee)`
FORTRAN stub to construct the atom list object.
- VEXTERNC void `Valist_dtor (Valist **thee)`
Destroys atom list object.
- VEXTERNC void `Valist_dtor2 (Valist *thee)`
FORTRAN stub to destroy atom list object.
- VEXTERNC Vrc_Codes `Valist_readPQR (Valist *thee, Vparam *param, Vio *sock)`
Fill atom list with information from a PQR file.
- VEXTERNC Vrc_Codes `Valist_readPDB (Valist *thee, Vparam *param, Vio *sock)`
Fill atom list with information from a PDB file.
- VEXTERNC Vrc_Codes `Valist_readXML (Valist *thee, Vparam *param, Vio *sock)`
Fill atom list with information from an XML file.
- VEXTERNC Vrc_Codes `Valist_getStatistics (Valist *thee)`
Load up Valist with various statistics.

8.10.1 Detailed Description

Container class for list of atom objects.

8.10.2 Function Documentation

8.10.2.1 VEXTERNC Valist* Valist_ctor()

Construct the atom list object.

Author

Nathan Baker

Returns

Pointer to newly allocated (empty) atom list

Definition at line 139 of file [valist.c](#).

Here is the call graph for this function:



8.10.2.2 VEXTERNC Vrc_Codes Valist_ctor2(Valist * thee)

FORTRAN stub to construct the atom list object.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Parameters

<i>thee</i>	Storage for new atom list
-------------	---------------------------

Definition at line 156 of file [valist.c](#).

Here is the caller graph for this function:



8.10.2.3 VEXTERN void Valist_dtor (Valist ***thee*)

Destroys atom list object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to storage for atom list
-------------	----------------------------------

Definition at line 168 of file [valist.c](#).

Here is the call graph for this function:



8.10.2.4 VEXTERN void Valist_dtor2 (Valist **thee*)

FORTRAN stub to destroy atom list object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to atom list object
-------------	-----------------------------

Definition at line 177 of file [valist.c](#).

Here is the caller graph for this function:

**8.10.2.5 VEXTERNC Vatom* Valist_getAtom (Valist * *thee*, int *i*)**

Get pointer to particular atom in list.

Author

Nathan Baker

Returns

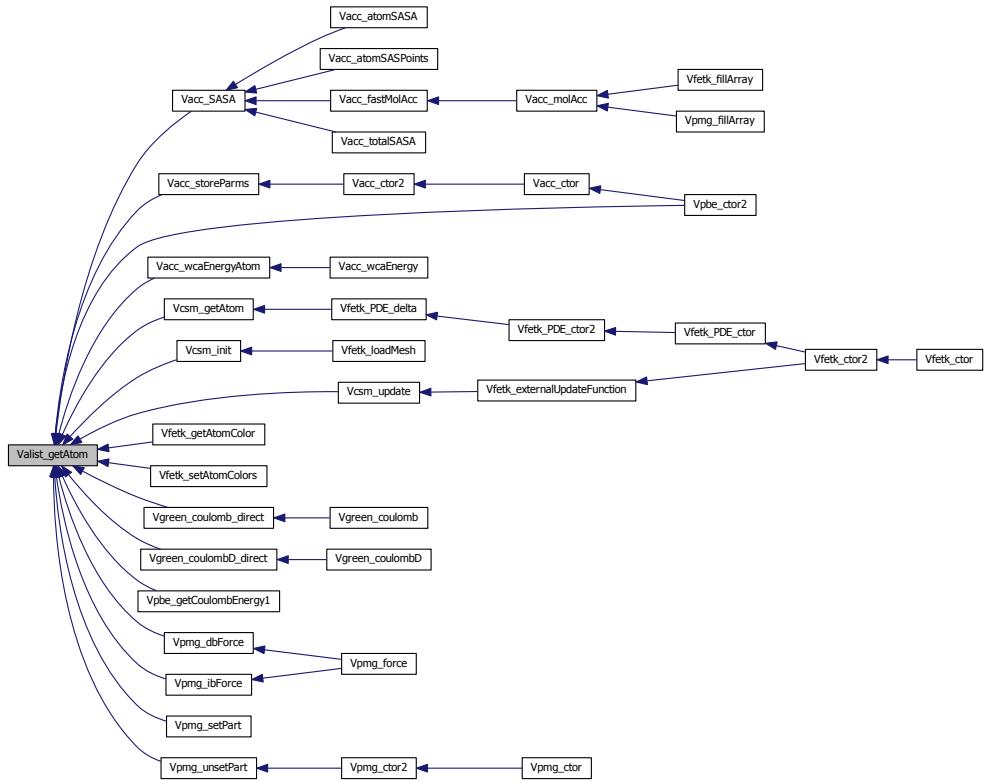
Pointer to atom object *i*

Parameters

<i>thee</i>	Atom list object
<i>i</i>	Index of atom in list

Definition at line 116 of file [valist.c](#).

Here is the caller graph for this function:



8.10.2.6 VEXTERNC `Vatom* Valist_getAtomList (Valist * thee)`

Get actual array of atom objects from the list.

Author

Nathan Baker

Returns

Array of atom objects

Parameters

<code>thee</code>	Atom list object
-------------------	------------------

Definition at line 96 of file [valist.c](#).

8.10.2.7 VEXTERNC double Valist_getCenterX (Valist * *thee*)

Get x-coordinate of molecule center.

Author

Nathan Baker

Returns

X-coordinate of molecule center

Parameters

<i>thee</i>	Atom list object
-------------	------------------

Definition at line 67 of file [valist.c](#).

8.10.2.8 VEXTERNC double Valist_getCenterY (Valist * *thee*)

Get y-coordinate of molecule center.

Author

Nathan Baker

Returns

Y-coordinate of molecule center

Parameters

<i>thee</i>	Atom list object
-------------	------------------

Definition at line 77 of file [valist.c](#).

8.10.2.9 VEXTERNC double Valist_getCenterZ (Valist * *thee*)

Get z-coordinate of molecule center.

Author

Nathan Baker

Returns

Z-coordinate of molecule center

Parameters

<i>thee</i>	Atom list object
-------------	------------------

Definition at line 86 of file [valist.c](#).

8.10.2.10 VEXTERNC int Valist_getNumberAtoms (Valist * *thee*)

Get number of atoms in the list.

Author

Nathan Baker

Returns

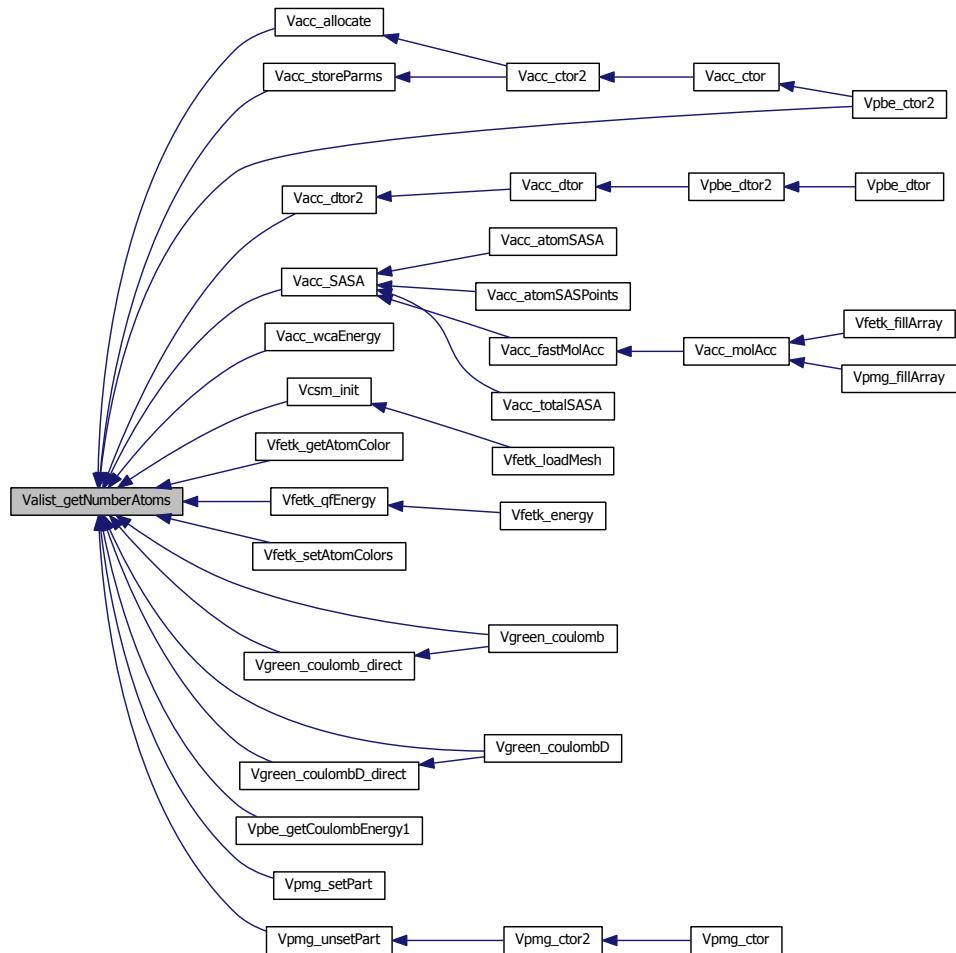
Number of atoms in list

Parameters

<i>thee</i>	Atom list object
-------------	------------------

Definition at line 106 of file [valist.c](#).

Here is the caller graph for this function:



8.10.2.11 VEXTERNC Vrc_Codes Valist_getStatistics (Valist * thee)

Load up Valist with various statistics.

Author

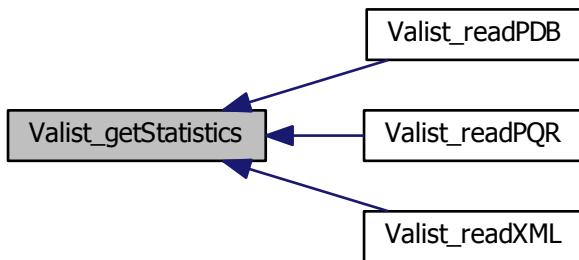
Nathan Baker, Yong Huang

Returns

Success enumeration

Definition at line 859 of file [valist.c](#).

Here is the caller graph for this function:

**8.10.2.12 VEXTERNC unsigned long int Valist_memChk (Valist * *thee*)**

Get total memory allocated for this object and its members.

Author

Nathan Baker

Returns

Total memory in bytes

Parameters

<i>thee</i>	Atom list object
-------------	------------------

Definition at line 130 of file [valist.c](#).

```
8.10.2.13 VEXTERNC Vrc_Codes Valist_readPDB ( Valist *thee, Vparam *param, Vio *sock
    )
```

Fill atom list with information from a PDB file.

Author

Nathan Baker, Todd Dolinsky, Yong Huang

Returns

Success enumeration

Note

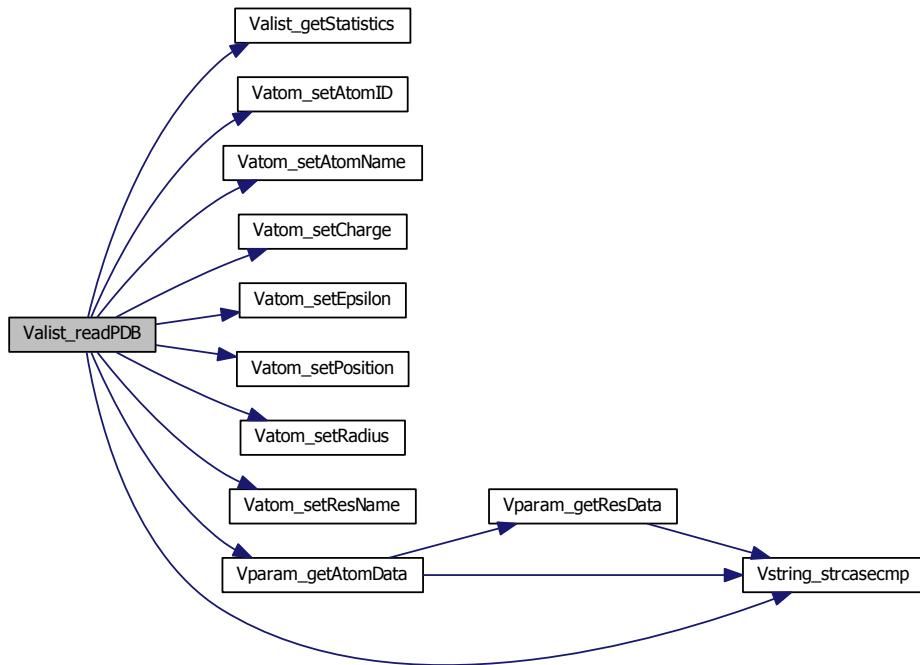
We don't actually respect PDB format; instead recognize whitespace- or tab-delimited fields which allows us to deal with structures with coordinates > 999 or < -999.

Parameters

<i>thee</i>	Atom list object
<i>param</i>	A pre-initialized parameter object
<i>sock</i>	Socket read for reading PDB file

Definition at line 516 of file [valist.c](#).

Here is the call graph for this function:



8.10.2.14 VEXTERNC Vrc_Codes Valist.readPQR (Valist * *thee*, Vparam * *param*, Vio * *sock*)

Fill atom list with information from a PQR file.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Note

- A PQR file has PDB structure with charge and radius in the last two columns instead of weight and occupancy

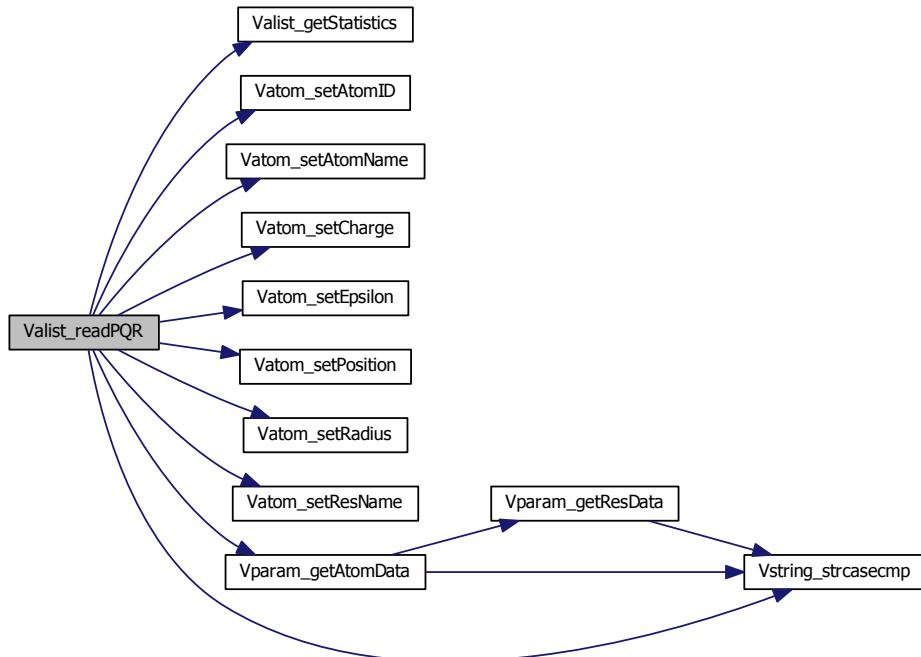
- We don't actually respect PDB format; instead recognize whitespace- or tab-delimited fields which allows us to deal with structures with coordinates > 999 or < -999.

Parameters

<i>thee</i>	Atom list object
<i>param</i>	A pre-initialized parameter object
<i>sock</i>	Socket reading for reading PQR file

Definition at line 607 of file [valist.c](#).

Here is the call graph for this function:



8.10.2.15 VEXTERNC Vrc_Codes Valist_readXML (Valist * *thee*, Vparam * *param*, Vio * *sock*)

Fill atom list with information from an XML file.

Author

Todd Dolinsky, Yong Huang

Returns

Success enumeration

Note

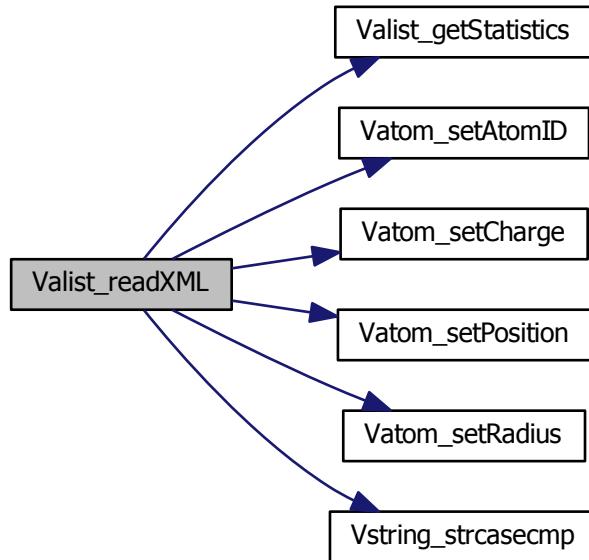
- The XML file must adhere to some guidelines, notably the presence of an <atom> tag with all other useful information (x, y, z, charge, and radius) as nested elements.

Parameters

<i>thee</i>	Atom list object
<i>param</i>	A pre-initialized parameter object
<i>sock</i>	Socket reading for reading PQR file

Definition at line [715](#) of file `valist.c`.

Here is the call graph for this function:



8.11 Vatom class

Atom class for interfacing APBS with PDB files.

Data Structures

- struct [sVatom](#)
Contains public data members for Vatom class/module.

Files

- file [vatom.h](#)
Contains declarations for class Vatom.
- file [vatom.c](#)

Class Vatom methods.

Defines

- #define `VMAX_RECLEN` 64

Residue name length.

TypeDefs

- typedef struct `sVatom` `Vatom`

Declaration of the Vatom class as the Vatom structure.

Functions

- VEXTERNC double * `Vatom_getPosition` (`Vatom` *thee)
Get atomic position.
- VEXTERNC void `Vatom_setRadius` (`Vatom` *thee, double radius)
Set atomic radius.
- VEXTERNC double `Vatom_getRadius` (`Vatom` *thee)
Get atomic position.
- VEXTERNC void `Vatom_setPartID` (`Vatom` *thee, int partID)
Set partition ID.
- VEXTERNC double `Vatom_getPartID` (`Vatom` *thee)
Get partition ID.
- VEXTERNC void `Vatom_setAtomID` (`Vatom` *thee, int id)
Set atom ID.
- VEXTERNC double `Vatom_getAtomID` (`Vatom` *thee)
Get atom ID.
- VEXTERNC void `Vatom_setCharge` (`Vatom` *thee, double charge)
Set atomic charge.
- VEXTERNC double `Vatom_getCharge` (`Vatom` *thee)
Get atomic charge.
- VEXTERNC void `Vatom_setEpsilon` (`Vatom` *thee, double epsilon)
Set atomic epsilon.
- VEXTERNC double `Vatom_getEpsilon` (`Vatom` *thee)
Get atomic epsilon.
- VEXTERNC unsigned long int `Vatom_memChk` (`Vatom` *thee)
Return the memory used by this structure (and its contents) in bytes.

- VEXTERNC void `Vatom_setResName` (`Vatom *thee`, `char resName[VMAX_RECLEN]`)

Set residue name.
- VEXTERNC void `Vatom_setAtomName` (`Vatom *thee`, `char atomName[VMAX_RECLEN]`)

Set atom name.
- VEXTERNC void `Vatom_getResName` (`Vatom *thee`, `char resName[VMAX_RECLEN]`)

Retrieve residue name.
- VEXTERNC void `Vatom_getAtomName` (`Vatom *thee`, `char atomName[VMAX_RECLEN]`)

Retrieve atom name.
- VEXTERNC `Vatom * Vatom_ctor` ()

Constructor for the Vatom class.
- VEXTERNC int `Vatom_ctor2` (`Vatom *thee`)

FORTRAN stub constructor for the Vatom class.
- VEXTERNC void `Vatom_dtor` (`Vatom **thee`)

Object destructor.
- VEXTERNC void `Vatom_dtor2` (`Vatom *thee`)

FORTRAN stub object destructor.
- VEXTERNC void `Vatom_setPosition` (`Vatom *thee`, `double position[3]`)

Set the atomic position.
- VEXTERNC void `Vatom_copyTo` (`Vatom *thee`, `Vatom *dest`)

Copy information to another atom.
- VEXTERNC void `Vatom_copyFrom` (`Vatom *thee`, `Vatom *src`)

Copy information to another atom.

8.11.1 Detailed Description

Atom class for interfacing APBS with PDB files.

8.11.2 Define Documentation

8.11.2.1 #define VMAX_RECLEN 64

Residue name length.

Author

Nathan Baker, David Gohara, Mike Schneiders

Definition at line 74 of file `vatom.h`.

8.11.3 Function Documentation

8.11.3.1 VEXTERNC void Vatom_copyFrom (Vatom * *thee*, Vatom * *src*)

Copy information to another atom.

Author

Nathan Baker

Parameters

<i>thee</i>	Destination for atom information
<i>src</i>	Source for atom information

Definition at line 175 of file [vatom.c](#).

Here is the call graph for this function:



8.11.3.2 VEXTERNC void Vatom_copyTo (Vatom * *thee*, Vatom * *dest*)

Copy information to another atom.

Author

Nathan Baker

Parameters

<i>thee</i>	Source for atom information
<i>dest</i>	Destination for atom information

Definition at line 166 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.3 VEXTERNC Vatom* Vatom_ctor()

Constructor for the Vatom class.

Author

Nathan Baker

Returns

Pointer to newly allocated Vatom object

Definition at line 131 of file [vatom.c](#).

Here is the call graph for this function:



8.11.3.4 VEXTERNC int Vatom_ctor2(Vatom *thee)

FORTRAN stub constructor for the Vatom class.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vatom allocated memory location
-------------	--

Returns

1 if successful, 0 otherwise

Definition at line 142 of file [vatom.c](#).

Here is the caller graph for this function:

**8.11.3.5 VEXTERNC void Vatom_dtor (Vatom ** *thee*)**

Object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of object to be destroyed
-------------	--

Definition at line 147 of file [vatom.c](#).

Here is the call graph for this function:



8.11.3.6 VEXTERNC void Vatom_dtor2 (Vatom * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
-------------	-----------------------------------

Definition at line 155 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.7 VEXTERNC double Vatom_getAtomID (Vatom * *thee*)

Get atom ID.

Author

Nathan Baker

Parameters

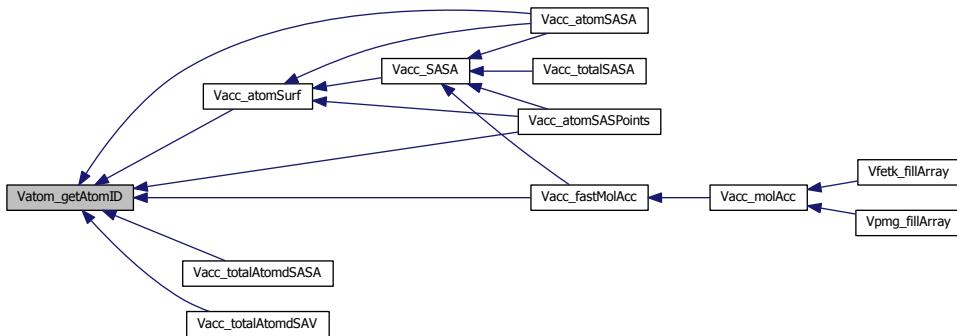
<i>thee</i>	Vatom object
-------------	--------------

Returns

Unique non-negative number

Definition at line 85 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.8 VEXTERNC void Vatom_getAtomName (Vatom * *thee*, char *atomName*[VMAX_RECLEN])

Retrieve atom name.

Author

Jason Wagoner

Parameters

<i>thee</i>	Vatom object
<i>atomName</i>	Atom name

Definition at line 203 of file [vatom.c](#).

8.11.3.9 VEXTERNC double Vatom_getCharge (Vatom * *thee*)

Get atomic charge.

Author

Nathan Baker

Parameters

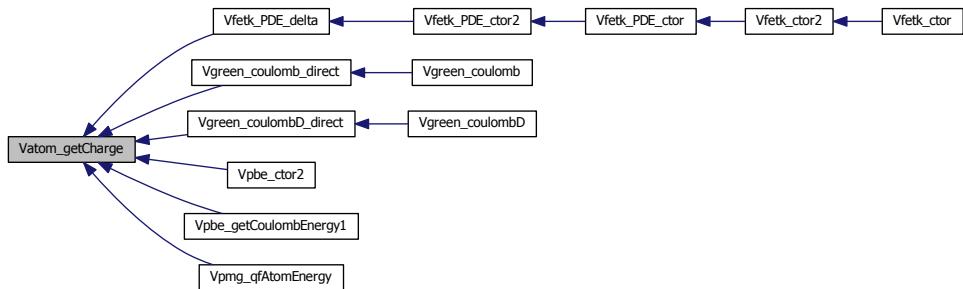
<i>thee</i>	Vatom object
-------------	--------------

Returns

Atom partial charge (in e)

Definition at line 120 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.10 VEXTERNC double Vatom_getEpsilon (Vatom * *thee*)

Get atomic epsilon.

Author

David Gohara

Parameters

<i>thee</i>	Vatom object
-------------	--------------

Returns

Atomic epsilon (in Å)

8.11.3.11 VEXTERNC double Vatom_getPartID (Vatom * *thee*)

Get partition ID.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
-------------	--------------

Returns

Partition ID; a negative value means this atom is not assigned to any partition

Definition at line 71 of file [vatom.c](#).

Here is the caller graph for this function:

**8.11.3.12 VEXTERNC double* Vatom_getPosition (Vatom * *thee*)**

Get atomic position.

Author

Nathan Baker

Parameters

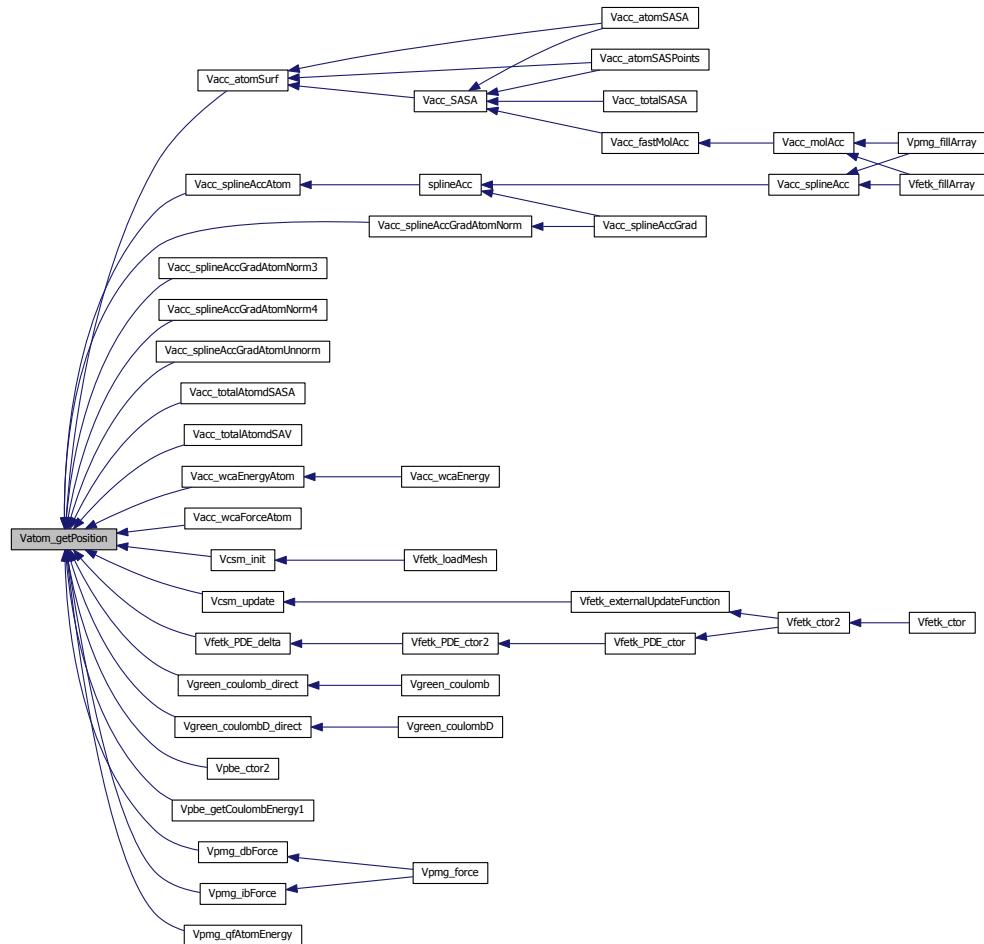
<i>thee</i>	Vatom object
-------------	--------------

Returns

Pointer to 3*double array of atomic coordinates (in Å)

Definition at line 64 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.13 VEXTERNC double Vatom_getRadius (Vatom * thee)

Get atomic position.

Author

Nathan Baker

Parameters

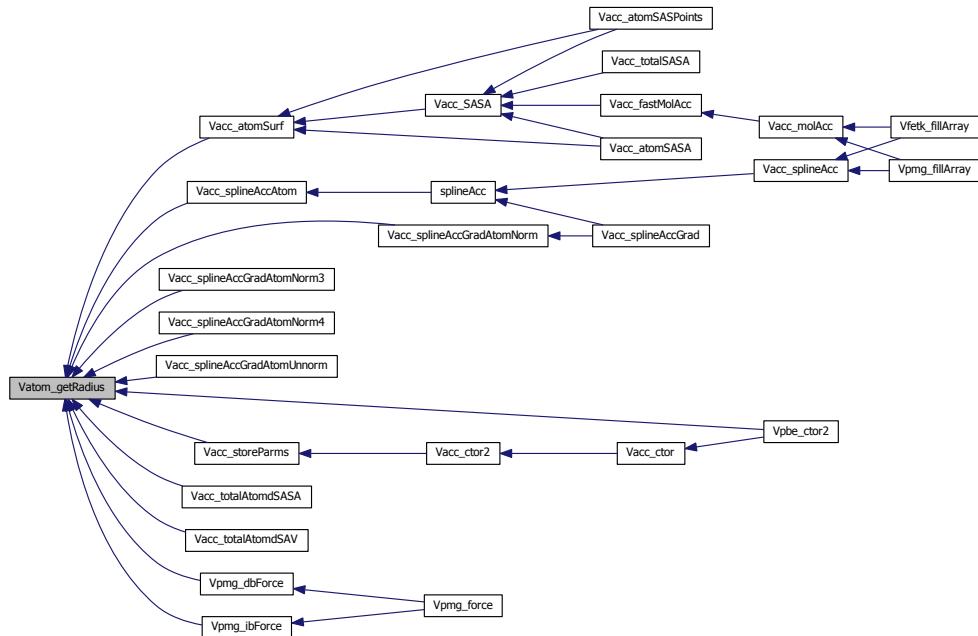
<i>thee</i>	Vatom object
-------------	--------------

Returns

Atomic radius (in Å)

Definition at line 106 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.14 VEXTERNC void Vatom_getResName (Vatom * *thee*, char *resName*[VMAX_RECLEN]
)

Retrieve residue name.

Author

Jason Wagoner

Parameters

<i>thee</i>	Vatom object
<i>resName</i>	Residue Name

Definition at line 188 of file [vatom.c](#).

8.11.3.15 VEXTERNC unsigned long int Vatom_memChk (Vatom * *thee*)

Return the memory used by this structure (and its contents) in bytes.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpmg object
-------------	-------------

Returns

The memory used by this structure and its contents in bytes

Definition at line 127 of file [vatom.c](#).

8.11.3.16 VEXTERNC void Vatom_setAtomID (Vatom * *thee*, int *id*)

Set atom ID.

Author

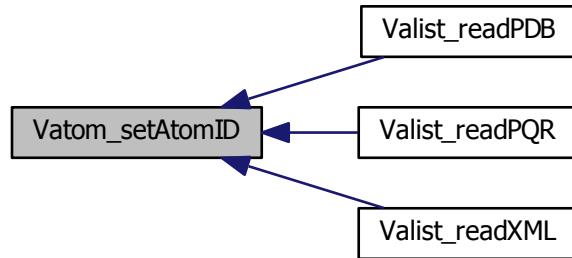
Nathan Baker

Parameters

<i>thee</i>	Vatom object
<i>id</i>	Unique non-negative number

Definition at line 92 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.17 VEXTERNC void Vatom_setAtomName (`Vatom * thee`, `char atomName[VMAX_RECLEN]`)

Set atom name.

Author

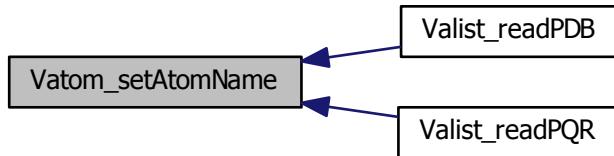
Jason Wagoner

Parameters

<code>thee</code>	Vatom object
<code>atomName</code>	Atom name

Definition at line 196 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.18 VEXTERNC void Vatom_setCharge (`Vatom * thee`, double `charge`)

Set atomic charge.

Author

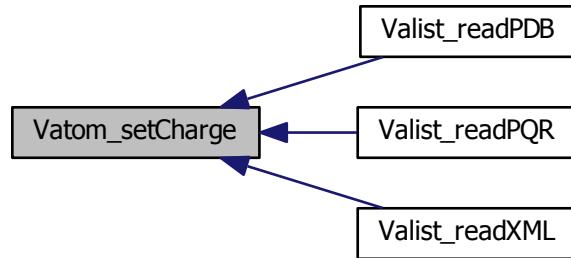
Nathan Baker

Parameters

<code>thee</code>	Vatom object
<code>charge</code>	Atom partial charge (in e)

Definition at line 113 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.19 VEXTERNC void Vatom_setEpsilon (Vatom * *thee*, double *epsilon*)

Set atomic epsilon.

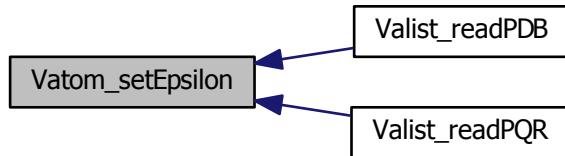
Author

David Gohara

Parameters

<i>thee</i>	Vatom object
<i>epsilon</i>	Atomic epsilon (in Å)

Here is the caller graph for this function:



8.11.3.20 VEXTERNC void Vatom_setPartID (Vatom * *thee*, int *partID*)

Set partition ID.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
<i>partID</i>	Partition ID; a negative value means this atom is not assigned to any partition

Definition at line 78 of file [vatom.c](#).

Here is the caller graph for this function:



8.11.3.21 VEXTERNC void Vatom_setPosition (Vatom * *thee*, double *position*[3])

Set the atomic position.

Author

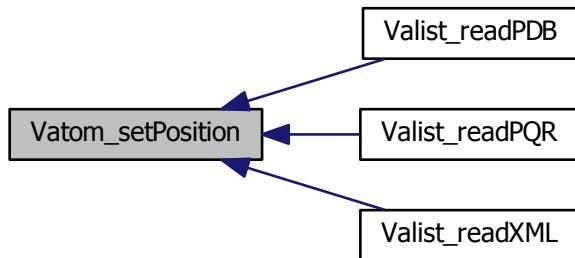
Nathan Baker

Parameters

<i>thee</i>	Vatom object to be modified
<i>position</i>	Coordinates (in Å)

Definition at line 157 of file [vatom.c](#).

Here is the caller graph for this function:

**8.11.3.22 VEXTERNC void Vatom_setRadius (Vatom * *thee*, double *radius*)**

Set atomic radius.

Author

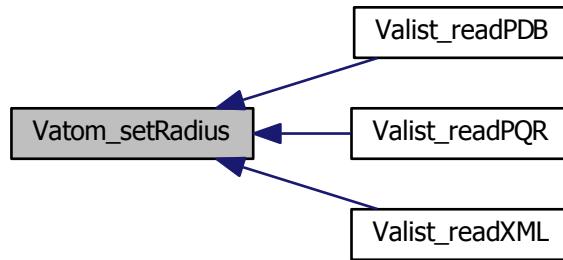
Nathan Baker

Parameters

<i>thee</i>	Vatom object
<i>radius</i>	Atomic radius (in Å)

Definition at line 99 of file [vatom.c](#).

Here is the caller graph for this function:



```
8.11.3.23 VEXTERNC void Vatom_setResName ( Vatom * thee, char resName[VMAX_RECLEN]  
)
```

Set residue name.

Author

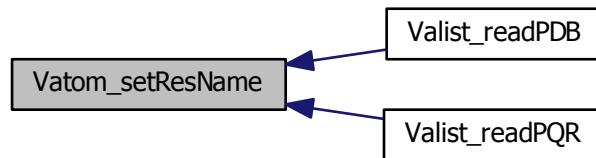
Jason Wagoner

Parameters

<i>thee</i>	Vatom object
<i>resName</i>	Residue Name

Definition at line 181 of file [vatom.c](#).

Here is the caller graph for this function:



8.12 Vcap class

Collection of routines which cap certain exponential and hyperbolic functions.

Files

- file [vcap.h](#)
Contains declarations for class Vcap.
- file [vcap.c](#)
Class Vcap methods.

Defines

- #define [EXPMAX](#) 85.00
Maximum argument for exp(), sinh(), or cosh()
- #define [EXPMIN](#) -85.00
Minimum argument for exp(), sinh(), or cosh()

Functions

- VEXTERNC double [Vcap_exp](#) (double x, int *ichop)
Provide a capped exp() function.
- VEXTERNC double [Vcap_sinh](#) (double x, int *ichop)
Provide a capped sinh() function.

- VEXTERNC double [Vcap_cosh](#) (double x, int *ichop)

Provide a capped cosh() function.

8.12.1 Detailed Description

Collection of routines which cap certain exponential and hyperbolic functions.

Note

These routines are based on FORTRAN code by Mike Holst

8.12.2 Function Documentation

8.12.2.1 VEXTERNC double [Vcap_cosh](#) (double x, int *ichop)

Provide a capped cosh() function.

If the argument x of [Vcap_cosh\(\)](#) exceeds EXPMAX or EXPMIN, then we return cosh(EXPMAX) or cosh(EXPMIN) rather than cosh(x).

Note

Original FORTRAN routine from PMG library by Mike Holst Original notes: to control overflow in the hyperbolic and exp functions, note that the following are the argument limits of the various functions on various machines after which overflow occurs: Convex C240, Sun 3/60, Sun SPARC, IBM RS/6000: sinh, cosh, exp: maximal argument (abs value) = 88.0d0 dsinh, dcosh, dexp: maximal argument (abs value) = 709.0d0

Author

Nathan Baker (based on FORTRAN code by Mike Holst)

Returns

cosh(x) or capped equivalent

Parameters

x	Argument to cosh()
ichop	Set to 1 if function capped, 0 otherwise

Definition at line 92 of file [vcap.c](#).

8.12.2 VEXTERNC double Vcap_exp (double x, int * ichop)

Provide a capped exp() function.

If the argument x of [Vcap_exp\(\)](#) exceeds EXPMAX or EXPMIN, then we return exp(EXPMAX) or exp(EXPMIN) rather than exp(x).

Note

Original FORTRAN routine from PMG library by Mike Holst Original notes: to control overflow in the hyperbolic and exp functions, note that the following are the argument limits of the various functions on various machines after which overflow occurs: Convex C240, Sun 3/60, Sun SPARC, IBM RS/6000: sinh, cosh, exp: maximal argument (abs value) = 88.0d0 dsinh, dcosh, dexp: maximal argument (abs value) = 709.0d0

Author

Nathan Baker (based on FORTRAN code by Mike Holst)

Returns

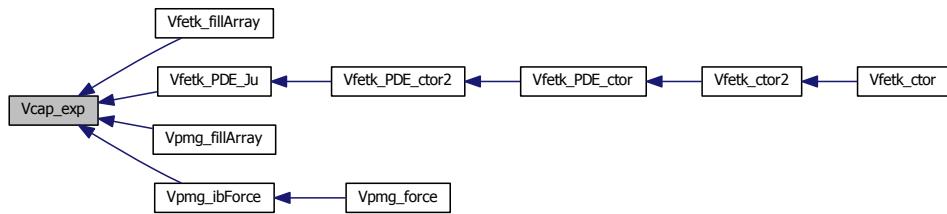
$\exp(x)$ or capped equivalent

Parameters

x	Argument to $\exp()$
ichop	Set to 1 if function capped, 0 otherwise

Definition at line 60 of file [vcap.c](#).

Here is the caller graph for this function:



8.12.2.3 VEXTERNC double Vcap_sinh (double x, int * ichop)

Provide a capped sinh() function.

If the argument x of [Vcap_sinh\(\)](#) exceeds EXPMAX or EXPMIN, then we return sinh(EXPMAX) or sinh(EXPMIN) rather than sinh(x).

Note

Original FORTRAN routine from PMG library by Mike Holst Original notes: to control overflow in the hyperbolic and exp functions, note that the following are the argument limits of the various functions on various machines after which overflow occurs: Convex C240, Sun 3/60, Sun SPARC, IBM RS/6000: sinh, cosh, exp: maximal argument (abs value) = 88.0d0 dsinh, dcosh, dexp: maximal argument (abs value) = 709.0d0

Author

Nathan Baker (based on FORTRAN code by Mike Holst)

Returns

$\sinh(x)$ or capped equivalent

Parameters

x	Argument to sinh()
ichop	Set to 1 if function capped, 0 otherwise

Definition at line 76 of file [vcap.c](#).

8.13 Vclist class

Atom cell list.

Data Structures

- struct [sVclistCell](#)

Atom cell list cell.
- struct [sVclist](#)

Atom cell list.

Files

- file [vclist.h](#)
Contains declarations for class Vclist.
- file [vclist.c](#)
Class Vclist methods.

Typedefs

- [typedef struct sVclistCell VclistCell](#)
Declaration of the VclistCell class as the VclistCell structure.
- [typedef struct sVclist Vclist](#)
Declaration of the Vclist class as the Vclist structure.
- [typedef enum eVclist_DomainMode Vclist_DomainMode](#)
Declaration of Vclist_DomainMode enumeration type.

Enumerations

- [enum eVclist_DomainMode { CLIST_AUTO_DOMAIN, CLIST_MANUAL_DOMAIN }](#)
Atom cell list domain setup mode.

Functions

- [VEXTERNC unsigned long int Vclist_memChk \(Vclist *thee\)](#)
Get number of bytes in this object and its members.
- [VEXTERNC double Vclist_maxRadius \(Vclist *thee\)](#)
Get the max probe radius value (in A) the cell list was constructed with.
- [VEXTERNC Vclist * Vclist_ctor \(Valist *alist, double max_radius, int npts\[VAPBS_-DIM\], Vclist_DomainMode mode, double lower_corner\[VAPBS_DIM\], double upper_corner\[VAPBS_DIM\]\)](#)
Construct the cell list object.
- [VEXTERNC Vrc_Codes Vclist_ctor2 \(Vclist *thee, Valist *alist, double max_radius, int npts\[VAPBS_DIM\], Vclist_DomainMode mode, double lower_corner\[VAPBS_-DIM\], double upper_corner\[VAPBS_DIM\]\)](#)
FORTRAN stub to construct the cell list object.
- [VEXTERNC void Vclist_dtor \(Vclist **thee\)](#)
Destroy object.
- [VEXTERNC void Vclist_dtor2 \(Vclist *thee\)](#)
FORTRAN stub to destroy object.

- VEXTERNC `VclistCell * Vclist_getCell (Vclist *thee, double position[VAPBS_DIM])`
Return cell corresponding to specified position or return VNULL.
- VEXTERNC `VclistCell * VclistCell_ctor (int natoms)`
Allocate and construct a cell list cell object.
- VEXTERNC `Vrc_Codes VclistCell_ctor2 (VclistCell *thee, int natoms)`
Construct a cell list object.
- VEXTERNC void `VclistCell_dtor (VclistCell **thee)`
Destroy object.
- VEXTERNC void `VclistCell_dtor2 (VclistCell *thee)`
FORTRAN stub to destroy object.

8.13.1 Detailed Description

Atom cell list.

8.13.2 Enumeration Type Documentation

8.13.2.1 enum eVclist_DomainMode

Atom cell list domain setup mode.

Author

Nathan Baker

Enumerator:

CLIST_AUTO_DOMAIN Setup the cell list domain automatically to encompass the entire molecule

CLIST_MANUAL_DOMAIN Specify the cell list domain manually through the constructor

Definition at line 79 of file [vclist.h](#).

8.13.3 Function Documentation

8.13.3.1 VEXTERNC `Vclist* Vclist_ctor (Valist * alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])`

Construct the cell list object.

Author

Nathan Baker

Returns

Newly allocated Vclist object

Parameters

<i>alist</i>	Molecule for cell list queries
<i>max_radius</i>	Max probe radius (\AA) to be queried
<i>npts</i>	Number of in hash table points in each direction
<i>mode</i>	Mode to construct table
<i>lower_-corner</i>	Hash table lower corner for manual construction (see mode variable); ignored otherwise
<i>upper_-corner</i>	Hash table upper corner for manual construction (see mode variable); ignored otherwise

Definition at line 80 of file [vclist.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.13.3.2 VEXTERNC Vrc_Codes Vclist_ctor2 (*Vclist * thee, Valist * alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]*)

FORTRAN stub to construct the cell list object.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Parameters

<i>thee</i>	Memory for Vclist objet
<i>alist</i>	Molecule for cell list queries
<i>max_radius</i>	Max probe radius (\AA) to be queried
<i>npts</i>	Number of in hash table points in each direction
<i>mode</i>	Mode to construct table
<i>lower_-corner</i>	Hash table lower corner for manual construction (see mode variable); ignored otherwise
<i>upper_-corner</i>	Hash table upper corner for manual construction (see mode variable); ignored otherwise

Definition at line 348 of file [vclist.c](#).

Here is the caller graph for this function:



8.13.3.3 VEXTERNC void Vclist_dtor (*Vclist ** thee*)

Destroy object.

Author

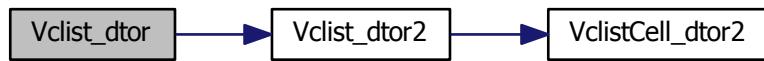
Nathan Baker

Parameters

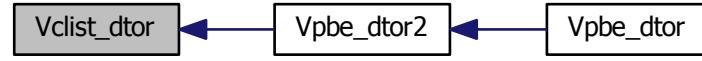
<i>thee</i>	Pointer to memory location of object
-------------	--------------------------------------

Definition at line [402](#) of file [vclist.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.13.3.4 VEXTERNC void Vclist_dtor2(Vclist * *thee*)

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object
-------------	-------------------

Definition at line 413 of file [vclist.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.13.3.5 VEXTERNC VclistCell* Vclist_getCell (Vclist * *thee*, double *position*[VAPBS_DIM])

Return cell corresponding to specified position or return VNULL.

Author

Nathan Baker

Returns

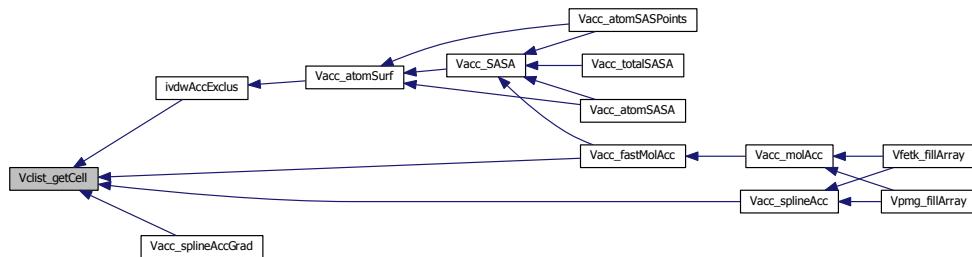
Pointer to VclistCell object or VNULL if no cell available (away from molecule).

Parameters

<i>thee</i>	Pointer to Vclist cell list
<i>position</i>	Position to evaluate

Definition at line 428 of file [vclist.c](#).

Here is the caller graph for this function:



8.13.3.6 VEXTERNC double Vclist_maxRadius (*Vclist * thee*)

Get the max probe radius value (in A) the cell list was constructed with.

Author

Nathan Baker

Returns

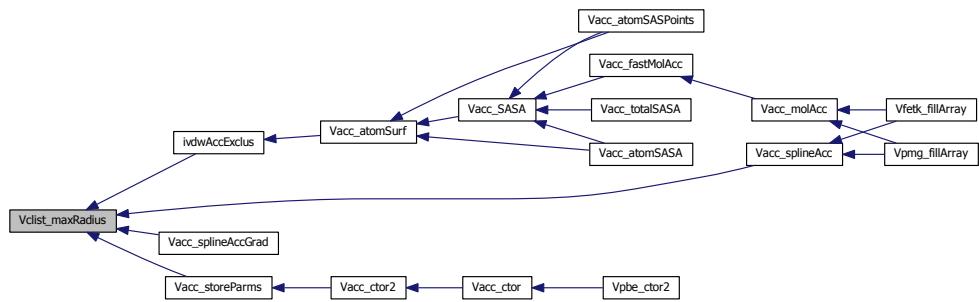
Max probe radius (in A)

Parameters

<i>thee</i>	Cell list object
-------------	------------------

Definition at line 73 of file [vclist.c](#).

Here is the caller graph for this function:



8.13.3.7 VEXTERNC unsigned long int Vclist_memChk (*Vclist * thee*)

Get number of bytes in this object and its members.

Author

Nathan Baker

Returns

Number of bytes allocated for object

Parameters

<i>thee</i>	Object for memory check
-------------	-------------------------

Definition at line 68 of file [vclist.c](#).

8.13.3.8 VEXTERNC *VclistCell** VclistCell_ctor (*int natoms*)

Allocate and construct a cell list cell object.

Author

Nathan Baker

Returns

Pointer to newly-allocated and constructed object.

Parameters

<i>natoms</i>	Number of atoms associated with this cell
---------------	---

Definition at line 454 of file [vclist.c](#).

Here is the call graph for this function:



8.13.3.9 VEXTERNC Vrc_Codes VclistCell_ctor2 (VclistCell * *thee*, int *natoms*)

Construct a cell list object.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Parameters

<i>thee</i>	Memory location for object
<i>natoms</i>	Number of atoms associated with this cell

Definition at line 466 of file [vclist.c](#).

Here is the caller graph for this function:



8.13.3.10 VEXTERNC void VclistCell_dtor (*VclistCell* ** *thee*)

Destroy object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of object
-------------	--------------------------------------

Definition at line 488 of file [vclist.c](#).

Here is the call graph for this function:



8.13.3.11 VEXTERNC void VclistCell_dtor2 (*VclistCell* * *thee*)

FORTRAN stub to destroy object.

Author

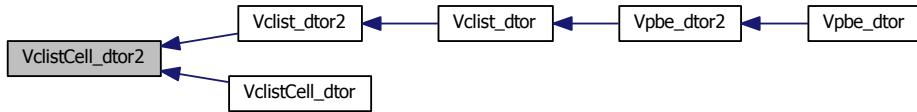
Nathan Baker

Parameters

<code>thee</code>	Pointer to object
-------------------	-------------------

Definition at line 499 of file `vclist.c`.

Here is the caller graph for this function:



8.14 Vgreen class

Provides capabilities for pointwise evaluation of free space Green's function for point charges in a uniform dielectric.

Data Structures

- struct `sVgreen`
Contains public data members for Vgreen class/module.

Files

- file `vgreen.h`
Contains declarations for class Vgreen.
- file `vgreen.c`
Class Vgreen methods.

Typedefs

- typedef struct `sVgreen` `Vgreen`
Declaration of the Vgreen class as the Vgreen structure.

Functions

- VEXTERNC `Valist * Vgreen_getValist (Vgreen *thee)`
Get the atom list associated with this Green's function object.
- VEXTERNC unsigned long int `Vgreen_memChk (Vgreen *thee)`
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC `Vgreen * Vgreen_ctor (Valist *alist)`
Construct the Green's function oracle.
- VEXTERNC int `Vgreen_ctor2 (Vgreen *thee, Valist *alist)`
FORTRAN stub to construct the Green's function oracle.
- VEXTERNC void `Vgreen_dtor (Vgreen **thee)`
Destruct the Green's function oracle.
- VEXTERNC void `Vgreen_dtor2 (Vgreen *thee)`
FORTRAN stub to destruct the Green's function oracle.
- VEXTERNC int `Vgreen_helmholtz (Vgreen *thee, int npos, double *x, double *y, double *z, double *val, double kappa)`
Get the Green's function for Helmholtz's equation integrated over the atomic point charges.
- VEXTERNC int `Vgreen_helmholtzD (Vgreen *thee, int npos, double *x, double *y, double *z, double *gradx, double *grady, double *gradz, double kappa)`
Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.
- VEXTERNC int `Vgreen_coulomb_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)`
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VEXTERNC int `Vgreen_coulomb (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)`
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)
- VEXTERNC int `Vgreen_coulombD_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)`
Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VEXTERNC int `Vgreen_coulombD (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)`
Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. Johnston/R. Krasny FMM library (if available)

8.14.1 Detailed Description

Provides capabilities for pointwise evaluation of free space Green's function for point charges in a uniform dielectric.

Note

Right now, these are very slow methods without any fast multipole acceleration.

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

8.14.2 Function Documentation

8.14.2.1 VEXTERNC int Vgreen_coulomb (*Vgreen * thee*, *int npos*, *double * x*, *double * y*,
*double * z*, *double * val*)

Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)

Returns the potential ϕ defined by

$$\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r . The potential is scaled to units of V.

Author

Nathan Baker

Parameters

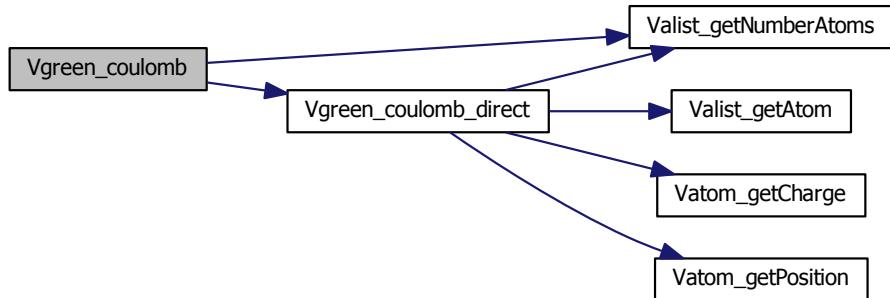
<i>thee</i>	Vgreen object
<i>npos</i>	The number of positions to evaluate
<i>x</i>	The npos x-coordinates
<i>y</i>	The npos y-coordinates
<i>z</i>	The npos z-coordinates
<i>val</i>	The npos values

Returns

1 if successful, 0 otherwise

Definition at line 259 of file [vgreen.c](#).

Here is the call graph for this function:



8.14.2.2 VEXTERNC int Vgreen.coulomb_direct (`Vgreen * thee`, `int npos`, `double * x`, `double * y`, `double * z`, `double * val`)

Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.

Returns the potential ϕ defined by

$$\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r . The potential is scaled to units of V.

Author

Nathan Baker

Parameters

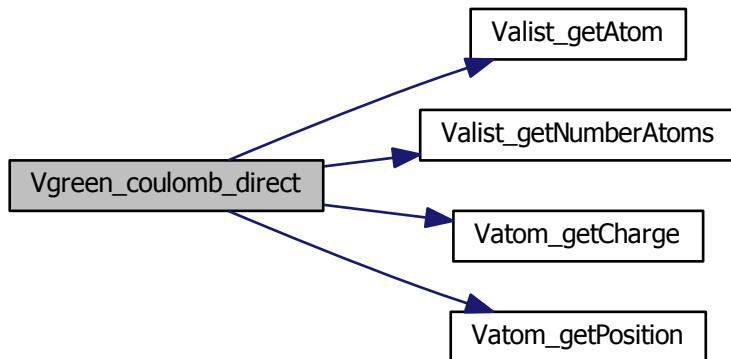
<code>thee</code>	Vgreen object
<code>npos</code>	The number of positions to evaluate
<code>x</code>	The npos x-coordinates
<code>y</code>	The npos y-coordinates
<code>z</code>	The npos z-coordinates
<code>val</code>	The npos values

Returns

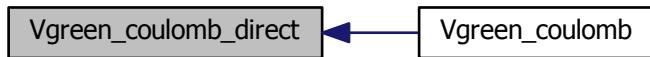
1 if successful, 0 otherwise

Definition at line 225 of file [vgreen.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



**8.14.2.3 VEXTERNC int Vgreen_coulombD (`Vgreen * thee`, `int npos`, `double * x`, `double * y`,
`double * z`, `double * pot`, `double * gradx`, `double * grady`, `double * gradz`)**

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. John-

ston/R. Krasny FMM library (if available)

Returns the field $\nabla\phi$ defined by

$$\nabla\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r . The field is scaled to units of V/Å.

Author

Nathan Baker

Parameters

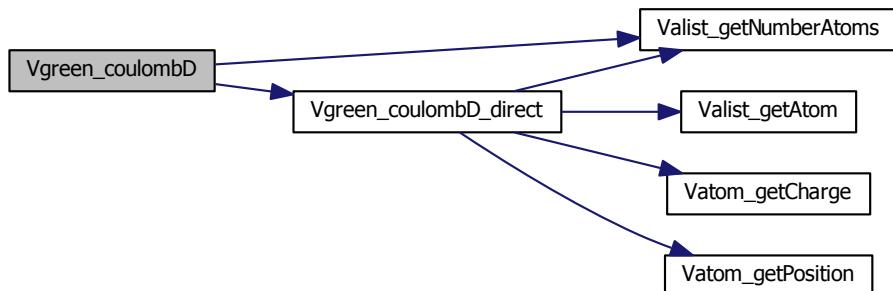
<i>thee</i>	Vgreen object
<i>npos</i>	The number of positions to evaluate
<i>x</i>	The <i>npos</i> x-coordinates
<i>y</i>	The <i>npos</i> y-coordinates
<i>z</i>	The <i>npos</i> z-coordinates
<i>pot</i>	The <i>npos</i> potential values
<i>gradx</i>	The <i>npos</i> gradient x-components
<i>grady</i>	The <i>npos</i> gradient y-components
<i>gradz</i>	The <i>npos</i> gradient z-components

Returns

1 if successful, 0 otherwise

Definition at line 363 of file [vgreen.c](#).

Here is the call graph for this function:



```
8.14.2.4 VEXTERNC int Vgreen_coulombD_direct ( Vgreen * thee, int npos, double * x,
double * y, double * z, double * pot, double * gradx, double * grady, double * gradz
)
```

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.

Returns the field $\nabla\phi$ defined by

$$\nabla\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r . The field is scaled to units of V/Å.

Author

Nathan Baker

Parameters

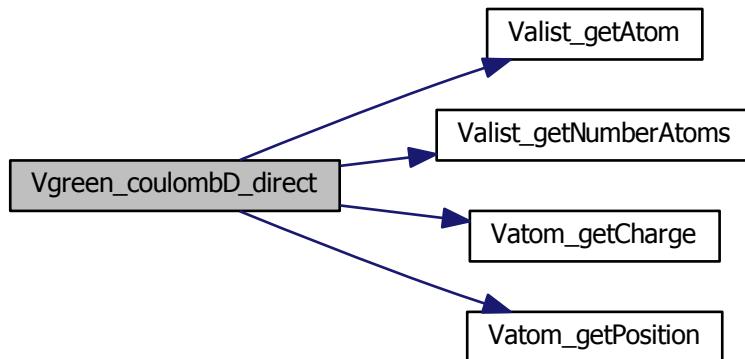
<i>thee</i>	Vgreen object
<i>npos</i>	The number of positions to evaluate
<i>x</i>	The npos x-coordinates
<i>y</i>	The npos y-coordinates
<i>z</i>	The npos z-coordinates
<i>pot</i>	The npos potential values
<i>gradx</i>	The npos gradient x-components
<i>grady</i>	The npos gradient y-components
<i>gradz</i>	The npos gradient z-components

Returns

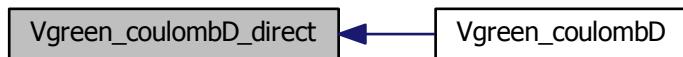
1 if successful, 0 otherwise

Definition at line 311 of file [vgreen.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.14.2.5 VEXTERNC `Vgreen*` `Vgreen_ctor (Valist * alist)`

Construct the Green's function oracle.

Author

Nathan Baker

Parameters

<code>alist</code>	Atom (charge) list associated with object
--------------------	---

Returns

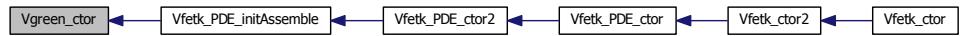
Pointer to newly allocated Green's function oracle

Definition at line 157 of file [vgreen.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.14.2.6 VEXTERNC int Vgreen_ctor2(Vgreen * *thee*, Valist * *alist*)

FORTRAN stub to construct the Green's function oracle.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory allocated for object
<i>alist</i>	Atom (charge) list associated with object

Returns

1 if successful, 0 otherwise

Definition at line 168 of file [vgreen.c](#).

Here is the caller graph for this function:



8.14.2.7 VEXTERNC void Vgreen_dtor (Vgreen ** *thee*)

Destruct the Green's function oracle.

Author

Nathan Baker

Parameters

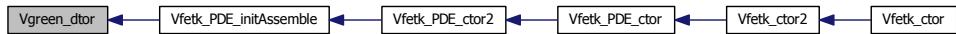
<i>thee</i>	Pointer to memory location for object
-------------	---------------------------------------

Definition at line 193 of file [vgreen.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.14.2.8 VEXTERNC void Vgreen_dtor2 (*Vgreen* * *thee*)

FORTRAN stub to destruct the Green's function oracle.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object
-------------	-------------------

Definition at line 201 of file [vgreen.c](#).

Here is the caller graph for this function:



8.14.2.9 VEXTERNC Valist* Vgreen_getValist (*Vgreen* * *thee*)

Get the atom list associated with this Green's function object.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgreen object
-------------	---------------

Returns

Pointer to Valist object associated with this Green's function object

Definition at line 143 of file [vgreen.c](#).

8.14.2.10 VEXTERNC int Vgreen_helmholtz (*Vgreen* * *thee*, int *npos*, double * *x*, double * *y*, double * *z*, double * *val*, double *kappa*)

Get the Green's function for Helmholtz's equation integrated over the atomic point charges.

Returns the potential ϕ defined by

$$\phi(r) = \sum_i \frac{q_i e^{-\kappa r_i}}{r_i}$$

where κ is the inverse screening length (in Å), q_i is the atomic charge (in e), and r_i is the distance from atom i to the observation point r . The potential is scaled to units of V.

Author

Nathan Baker

Bug

Not implemented yet

Note

Not implemented yet

Parameters

<i>thee</i>	Vgreen object
<i>npos</i>	Number of positions to evaluate
<i>x</i>	The <i>npos</i> x-coordinates
<i>y</i>	The <i>npos</i> y-coordinates
<i>z</i>	The <i>npos</i> z-coordinates
<i>val</i>	The <i>npos</i> values
<i>kappa</i>	The value of κ (see above)

Returns

1 if successful, 0 otherwise

Definition at line 210 of file [vgreen.c](#).

8.14.2.11 VEXTERNC int Vgreen_helmholtzD (Vgreen * *thee*, int *npos*, double * *x*, double * *y*, double * *z*, double * *gradx*, double * *grady*, double * *gradz*, double *kappa*)

Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.

Returns the field $\nabla\phi$ defined by

$$\nabla\phi(r) = \nabla \sum_i \frac{q_i e^{-\kappa r_i}}{r_i}$$

where κ is the inverse screening length (in Å). q_i is the atomic charge (in e), and r_i ; r_{-i} is the distance from atom i to the observation point r . The potential is scaled to units of V/Å.

Author

Nathan Baker

Bug

Not implemented yet

Note

Not implemented yet

Parameters

<i>thee</i>	Vgreen object
<i>npos</i>	The number of positions to evaluate
<i>x</i>	The npos x-coordinates
<i>y</i>	The npos y-coordinates
<i>z</i>	The npos z-coordinates
<i>gradx</i>	The npos gradient x-components
<i>grady</i>	The npos gradient y-components
<i>gradz</i>	The npos gradient z-components
<i>kappa</i>	The value of κ (see above)

Returns

int 1 if sucessful, 0 otherwise

Definition at line 217 of file [vgreen.c](#).

8.14.2.12 VEXTERNC unsigned long int Vgreen.memChk (Vgreen * *thee*)

Return the memory used by this structure (and its contents) in bytes.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgreen object
-------------	---------------

Returns

The memory used by this structure and its contents in bytes

Definition at line 150 of file [vgreen.c](#).

8.15 Vhal class

A "class" which consists solely of macro definitions which are used by several other classes.

Files

- file [vhal.h](#)

Contains generic macro definitions for APBS.

Defines

- `#define APBS_TIMER_WALL_CLOCK 26`
APBS total execution timer ID.
- `#define APBS_TIMER_SETUP 27`
APBS setup timer ID.
- `#define APBS_TIMER_SOLVER 28`
APBS solver timer ID.
- `#define APBS_TIMER_ENERGY 29`
APBS energy timer ID.
- `#define APBS_TIMER_FORCE 30`
APBS force timer ID.
- `#define APBS_TIMER_TEMP1 31`
APBS temp timer #1 ID.
- `#define APBS_TIMER_TEMP2 32`
APBS temp timer #2 ID.
- `#define MAXMOL 5`
The maximum number of molecules that can be involved in a single PBE calculation.
- `#define MAXION 10`
The maximum number of ion species that can be involved in a single PBE calculation.
- `#define MAXFOCUS 5`
The maximum number of times an MG calculation can be focused.
- `#define VMGNLEV 4`

- `#define VREDFRAC 0.25`

Minimum number of levels in a multigrid calculations.
- `#define VAPBS_NVS 4`

Maximum reduction of grid spacing during a focusing calculation.
- `#define VAPBS_DIM 3`

Number of vertices per simplex (hard-coded to 3D)
- `#define VAPBS_RIGHT 0`

Our dimension.
- `#define VAPBS_UP 2`

Face definition for a volume.
- `#define VAPBS_LEFT 3`

Face definition for a volume.
- `#define VAPBS_BACK 4`

Face definition for a volume.
- `#define VAPBS_DOWN 5`

Face definition for a volume.
- `#define VPMGSMALL 1e-12`

A small number used in Vpmg to decide if points are on/off grid-lines or non-zero (etc.)
- `#define SINH_MIN -85.0`

Used to set the min values acceptable for sinh chopping.
- `#define SINH_MAX 85.0`

Used to set the max values acceptable for sinh chopping.
- `#define VF77_MANGLE(name, NAME) name`

Name-mangling macro for using FORTRAN functions in C code.
- `#define VFLOOR(value) floor(value)`

Wrapped floor to fix floating point issues in the Intel compiler.
- `#define VEMBED(rctag)`

Allows embedding of RCS ID tags in object files.

TypeDefs

- `typedef enum eVhal_PBEType Vhal_PBEType`

Declaration of the Vhal_PBEType type as the Vhal_PBEType enum.
- `typedef enum eVhal_IPKEYType Vhal_IPKEYType`

Declaration of the Vhal_IPKEYType type as the Vhal_IPKEYType enum.

- **typedef enum eVhal_NONLINType Vhal_NONLINType**

Declaration of the Vhal_NONLINType type as the Vhal_NONLINType enum.

- **typedef enum eVoutput_Format Voutput_Format**

Declaration of the Voutput_Format type as the VOutput_Format enum.

- **typedef enum eVbcfl Vbcfl**

Declare Vbcfl type.

- **typedef enum eVsurf_Meth Vsurf_Meth**

Declaration of the Vsurf_Meth type as the Vsurf_Meth enum.

- **typedef enum eVchrg_Meth Vchrg_Meth**

Declaration of the Vchrg_Meth type as the Vchrg_Meth enum.

- **typedef enum eVchrg_Src Vchrg_Src**

Declaration of the Vchrg_Src type as the Vchrg_Meth enum.

- **typedef enum eVdata_Type Vdata_Type**

Declaration of the Vdata_Type type as the Vdata_Type enum.

- **typedef enum eVdata_Format Vdata_Format**

Declaration of the Vdata_Format type as the Vdata_Format enum.

Enumerations

- **enum eVrc_Codes { VRC_WARNING = -1, VRC_FAILURE = 0, VRC_SUCCESS = 1 }**

Return code enumerations.

- **enum eVsol_Meth {**

VSOL_CGMG, VSOL_Newton, VSOL_MG, VSOL(CG,

VSOLSOR, VSOL_RBGS, VSOL_WJ, VSOL_Richardson,

VSOL_CGMGAqua, VSOL_NewtonAqua }

Solution Method enumerations.

- **enum eVsurf_Meth {**

VSM_MOL = 0, VSM_MOLSMOOTH = 1, VSM_SPLINE = 2, VSM_SPLINE3 = 3,

VSM_SPLINE4 = 4 }

Types of molecular surface definitions.

- **enum eVhal_PBEType {**

PBE_LPBE, PBE_NPBE, PBE_LRPBE, PBE_NRPBE,

PBE_SMPBE }

Version of PBE to solve.

- **enum eVhal_IPKEYType { IPKEY_SMPBE = -2, IPKEY_LPBE, IPKEY_NPBE }**

Type of ipkey to use for MG methods.

- enum eVhal_NONLINType {
 NONLIN_LPBE = 0, **NONLIN_NPBE**, **NONLIN_SMPBE**, **NONLIN_LPBEAQUA**,
NONLIN_NPBEAQUA }

Type of nonlinear to use for MG methods.
- enum eVoutput_Format { **OUTPUT_NULL**, **OUTPUT_FLAT** }

Output file format.
- enum eVbcfl {
 BCFL_ZERO = 0, **BCFL_SDH** = 1, **BCFL_MDH** = 2, **BCFL_UNUSED** = 3,
BCFL_FOCUS = 4, **BCFL_MEM** = 5, **BCFL_MAP** = 6 }

Types of boundary conditions.
- enum eVchrg_Meth { **VCM_TRI** = 0, **VCM_BSPL2** = 1, **VCM_BSPL4** = 2 }

Types of charge discretization methods.
- enum eVchrg_Src { **VCM_CHARGE** = 0, **VCM_PERMANENT** = 1, **VCM_INDUCED** = 2, **VCM_NLINDUCED** = 3 }

Charge source.
- enum eVdata_Type {
 VDT_CHARGE, **VDT_POT**, **VDT_ATOMPOT**, **VDT_SMOL**,
VDT_SSPL, **VDT_VDW**, **VDT_IVDW**, **VDT_LAP**,
VDT_EDENS, **VDT_NDENS**, **VDT_QDENS**, **VDT_DIELX**,
VDT_DIELY, **VDT_DIELZ**, **VDT_KAPPA** }

Types of (scalar) data that can be written out of APBS.
- enum eVdata_Format {
 VDF_DX = 0, **VDF_UHBD** = 1, **VDF_AVIS** = 2, **VDF_MCSF** = 3,
VDF_GZ = 4, **VDF_FLAT** = 5 }

Format of data for APBS I/O.

8.15.1 Detailed Description

A "class" which consists solely of macro definitions which are used by several other classes.

8.15.2 Define Documentation

8.15.2.1 #define MAX_SPHERE PTS 50000

Maximum number of points on a sphere.

Note

Used by VaccSurf

Definition at line [420](#) of file `vhal.h`.

8.15.2.2 #define VAPBS_BACK 4

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [444](#) of file `vhal.h`.

8.15.2.3 #define VAPBS_DOWN 5

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [450](#) of file `vhal.h`.

8.15.2.4 #define VAPBS_FRONT 1

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [426](#) of file `vhal.h`.

8.15.2.5 #define VAPBS_LEFT 3

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [438](#) of file `vhal.h`.

8.15.2.6 #define VAPBS_RIGHT 0

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [414](#) of file [vhal.h](#).

8.15.2.7 #define VAPBS_UP 2

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [432](#) of file [vhal.h](#).

8.15.2.8 #define VEMBED(*rctag*)

Value:

```
VPRIVATE const char* rctag; \
    static void* use_rcsid=(0 ? &use_rcsid : (void**) &rcsid);
```

Allows embedding of RCS ID tags in object files.

Author

Mike Holst

Definition at line [569](#) of file [vhal.h](#).

8.15.2.9 #define VFLOOR(*value*) floor(*value*)

Wrapped floor to fix floating point issues in the Intel compiler.

Author

Todd Dolinksy

Definition at line [560](#) of file [vhal.h](#).

8.15.3 Enumeration Type Documentation

8.15.3.1 enum eVbcfl

Types of boundary conditions.

Author

Nathan Baker

Enumerator:

BCFL_ZERO Zero Dirichlet boundary conditions

BCFL_SDH Single-sphere Debye-Huckel Dirichlet boundary condition

BCFL_MDH Multiple-sphere Debye-Huckel Dirichlet boundary condition

BCFL_UNUSED Unused boundary condition method (placeholder)

BCFL_FOCUS Focusing Dirichlet boundary condition

BCFL_MEM Focusing membrane boundary condition

BCFL_MAP Skip first level of focusing use an external map

Definition at line 214 of file [vhal.h](#).

8.15.3.2 enum eVchrg_Meth

Types of charge discretization methods.

Author

Nathan Baker

Enumerator:

VCM_TRI1 Trilinear interpolation of charge to 8 nearest grid points. The traditional method; not particularly good to use with PBE forces.

VCM_BSPL2 Cubic B-spline across nearest- and next-nearest-neighbors. Mainly for use in grid-sensitive applications (such as force calculations).

VCM_BSPL4 5th order B-spline for AMOEBA permanent multipoles.

Definition at line 237 of file [vhal.h](#).

8.15.3.3 enum eVchrg_Src

Charge source.

Author

Michael Schnieders

Enumerator:

VCM_CHARGE Partial Charge source distribution

VCM_PERMANENT Permanent Multipole source distribution

VCM_INDUCED Induced Dipole source distribution

VCM_NLINDUCED NL Induced Dipole source distribution

Definition at line [258](#) of file [vhal.h](#).

8.15.3.4 enum eVdata_Format

Format of data for APBS I/O.

Author

Nathan Baker

Enumerator:

VDF_DX OpenDX (Data Explorer) format

VDF_UHBD UHBD format

VDF_AVIS AVS UCD format

VDF_MCSF FEtk MC Simplex Format (MCSF)

VDF_GZ Binary file (GZip)

VDF_FLAT Write flat file

Definition at line [316](#) of file [vhal.h](#).

8.15.3.5 enum eVdata_Type

Types of (scalar) data that can be written out of APBS.

Author

Nathan Baker

Enumerator:

VDT_CHARGE Charge distribution (e)

VDT_POT Potential (kT/e)

VDT_ATOMPOT Atom potential (kT/e)

VDT_SMOL Solvent accessibility defined by molecular/Connolly surface definition (1 = accessible, 0 = inaccessible)

VDT_SSPL Spline-based solvent accessibility (1 = accessible, 0 = inaccessible)

VDT_VDW van der Waals-based accessibility (1 = accessible, 0 = inaccessible)

VDT_IVDW Ion accessibility/inflated van der Waals (1 = accessible, 0 = inaccessible)

VDT_LAP Laplacian of potential ($kT/e/A^2$)

VDT_EDENS Energy density $\epsilon(\nabla u)^2$, where u is potential ($kT/e/A^2$)

VDT_NDENS Ion number density $\sum c_i \exp(-q_i u)^2$, where u is potential (output in M)

VDT_QDENS Ion charge density $\sum q_i c_i \exp(-q_i u)^2$, where u is potential (output in $e_c M$)

VDT_DIELX Dielectric x-shifted map as calculated with the currently specified scheme (dimensionless)

VDT_DIELY Dielectric y-shifted map as calculated with the currently specified scheme (dimensionless)

VDT_DIELZ Dielectric z-shifted map as calculated with the currently specified scheme (dimensionless)

VDT_KAPPA Kappa map as calculated with the currently specified scheme (m^{-3})

Definition at line 276 of file [vhal.h](#).

8.15.3.6 enum eVhal_IPKEYType

Type of ipkey to use for MG methods.

Enumerator:

IPKEY_SMPBE SMPBE ipkey

IPKEY_LPBE LPBE ipkey

IPKEY_NPBE NPBE ipkey

Definition at line 164 of file [vhal.h](#).

8.15.3.7 enum eVhal_PBEType

Version of PBE to solve.

Enumerator:

- PBE_LPBE** Traditional Poisson-Boltzmann equation, linearized
- PBE_NPBE** Traditional Poisson-Boltzmann equation, full
- PBE_LRPBE** Regularized Poisson-Boltzmann equation, linearized
- PBE_SMPBE** < Regularized Poisson-Boltzmann equation, full SM PBE

Definition at line 146 of file [vhal.h](#).

8.15.3.8 enum eVoutput_Format

Output file format.

Enumerator:

- OUTPUT_NULL** No output
- OUTPUT_FLAT** Output in flat-file format

Definition at line 198 of file [vhal.h](#).

8.15.3.9 enum eVrc_Codes

Return code enumerations.

Author

David Gohara

Note

Note that the enumerated values are opposite the standard for FAILURE and SUCCESS

Enumerator:

- VRC_FAILURE** A non-fatal error
- VRC_SUCCESS** A fatal error

Definition at line 73 of file [vhal.h](#).

8.15.3.10 enum eVsol_Meth

Solution Method enumerations.

Author

David Gohara

Note

Note that the enumerated values are opposite the standard for FAILURE and SUCCESS

Definition at line 88 of file [vhal.h](#).

8.15.3.11 enum eVsurf_Meth

Types of molecular surface definitions.

Author

Nathan Baker

Enumerator:

VSM_MOL Ion accessibility is defined using inflated van der Waals radii, the dielectric coefficient () is defined using the molecular (Conolly) surface definition without smoothing

VSM_MOLSMOOTH As VSM_MOL but with a simple harmonic average smoothing

VSM_SPLINE Spline-based surface definitions. This is primarily for use with force calculations, since it requires substantial reparameterization of radii. This is based on the work of Im et al, Comp. Phys. Comm. 111 , (1998) and uses a cubic spline to define a smoothly varying characteristic function for the surface-based parameters. Ion accessibility is defined using inflated van der Waals radii with the spline function and the dielectric coefficient is defined using the standard van der Waals radii with the spline function.

VSM_SPLINE3 A 5th order polynomial spline is used to create a smoothly varying characteristic function (continuity through 2nd derivatives) for surface based paramters.

VSM_SPLINE4 A 7th order polynomial spline is used to create a smoothly varying characteristic function (continuity through 3rd derivatives) for surface based paramters.

Definition at line 109 of file [vhal.h](#).

8.16 Vparam class

Reads and assigns charge/radii parameters.

Data Structures

- struct **sVparam_AtomData**
AtomData sub-class; stores atom data.
- struct **Vparam_ResData**
ResData sub-class; stores residue data.
- struct **Vparam**
Reads and assigns charge/radii parameters.

Files

- file **vparam.h**
*Contains declarations for class **Vparam**.*
- file **vparam.c**
*Class **Vparam** methods.*

Typedefs

- typedef struct **sVparam_AtomData Vparam_AtomData**
*Declaration of the **Vparam_AtomData** class as the **sVparam_AtomData** structure.*
- typedef struct **Vparam_ResData Vparam_ResData**
*Declaration of the **Vparam_ResData** class as the **Vparam_ResData** structure.*
- typedef struct **Vparam Vparam**
*Declaration of the **Vparam** class as the **Vparam** structure.*

Functions

- VEXTERNC unsigned long int **Vparam_memChk** (**Vparam** *thee)
Get number of bytes in this object and its members.
- VEXTERNC **Vparam_AtomData** * **Vparam_AtomData_ctor** ()
Construct the object.
- VEXTERNC int **Vparam_AtomData_ctor2** (**Vparam_AtomData** *thee)
FORTRAN stub to construct the object.
- VEXTERNC void **Vparam_AtomData_dtor** (**Vparam_AtomData** **thee)
Destroy object.
- VEXTERNC void **Vparam_AtomData_dtor2** (**Vparam_AtomData** *thee)
FORTRAN stub to destroy object.
- VEXTERNC void **Vparam_AtomData_copyTo** (**Vparam_AtomData** *thee, **Vparam_AtomData** *dest)

Copy current atom object to destination.

- VEXTERNC void [Vparam_ResData_copyTo](#) ([Vparam_ResData](#) *thee, [Vparam_ResData](#) *dest)
Copy current residue object to destination.
- VEXTERNC void [Vparam_AtomData_copyFrom](#) ([Vparam_AtomData](#) *thee, [Vparam_AtomData](#) *src)
Copy current atom object from another.
- VEXTERNC [Vparam_ResData](#) * [Vparam_ResData_ctor](#) ([Vmem](#) *mem)
Construct the object.
- VEXTERNC int [Vparam_ResData_ctor2](#) ([Vparam_ResData](#) *thee, [Vmem](#) *mem)

FORTRAN stub to construct the object.

- VEXTERNC void [Vparam_ResData_dtor](#) ([Vparam_ResData](#) **thee)
Destroy object.
- VEXTERNC void [Vparam_ResData_dtor2](#) ([Vparam_ResData](#) *thee)
FORTRAN stub to destroy object.
- VEXTERNC [Vparam](#) * [Vparam_ctor](#) ()
Construct the object.
- VEXTERNC int [Vparam_ctor2](#) ([Vparam](#) *thee)
FORTRAN stub to construct the object.
- VEXTERNC void [Vparam_dtor](#) ([Vparam](#) **thee)
Destroy object.
- VEXTERNC void [Vparam_dtor2](#) ([Vparam](#) *thee)
FORTRAN stub to destroy object.
- VEXTERNC [Vparam_ResData](#) * [Vparam_getResData](#) ([Vparam](#) *thee, char resName[VMAX_ARGLEN])
Get residue data.
- VEXTERNC [Vparam_AtomData](#) * [Vparam_getAtomData](#) ([Vparam](#) *thee, char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN])
Get atom data.
- VEXTERNC int [Vparam_readFlatFile](#) ([Vparam](#) *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)
Read a flat-file format parameter database.
- VEXTERNC int [Vparam_readXMLFile](#) ([Vparam](#) *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)
Read an XML format parameter database.
- VPRIVATE int [readFlatFileLine](#) ([Vio](#) *sock, [Vparam_AtomData](#) *atom)
Read a single line of the flat file database.
- VPRIVATE int [readXMLFileAtom](#) ([Vio](#) *sock, [Vparam_AtomData](#) *atom)
Read atom information from an XML file.

Variables

- VPRIVATE char * **MCwhiteChars** = " =;,\t\n\r"
Whitespace characters for socket reads.
- VPRIVATE char * **MCcommChars** = "#%"
Comment characters for socket reads.
- VPRIVATE char * **MCxmlwhiteChars** = " =;,\t\n<>"
Whitespace characters for XML socket reads.

8.16.1 Detailed Description

Reads and assigns charge/radii parameters.

8.16.2 Function Documentation

8.16.2.1 VPRIVATE int readFlatFileLine (**Vio * sock**, **Vparam_AtomData * atom**)

Read a single line of the flat file database.

Author

Nathan Baker

Parameters

sock	Socket ready for reading
atom	Atom to hold parsed data

Returns

1 if successful, 0 otherwise

Definition at line 696 of file [vparam.c](#).

Here is the caller graph for this function:



8.16.2.2 VPRIVATE int readXMLFileAtom (Vio * *sock*, Vparam_AtomData * *atom*)

Read atom information from an XML file.

Author

Todd Dolinsky

Parameters

<i>sock</i>	Socket ready for reading
<i>atom</i>	Atom to hold parsed data

Returns

1 if successful, 0 otherwise

Definition at line [615](#) of file [vparam.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
8.16.2.3 VEXTERNC void Vparam_AtomData_copyFrom ( Vparam_AtomData * thee,  
Vparam_AtomData * src )
```

Copy current atom object from another.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to destination object
<i>src</i>	Pointer to source object

Definition at line [612](#) of file [vparam.c](#).

Here is the call graph for this function:



```
8.16.2.4 VEXTERNC void Vparam_AtomData_copyTo ( Vparam_AtomData * thee,  
Vparam_AtomData * dest )
```

Copy current atom object to destination.

Author

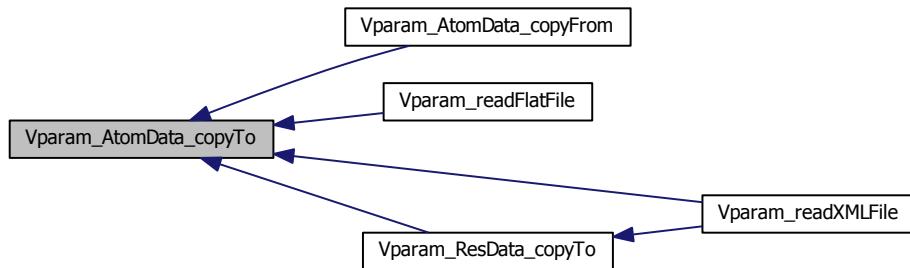
Nathan Baker

Parameters

<i>thee</i>	Pointer to source object
<i>dest</i>	Pointer to destination object

Definition at line [576](#) of file [vparam.c](#).

Here is the caller graph for this function:



8.16.2.5 VEXTERNC Vparam_AtomData* Vparam_AtomData.ctor()

Construct the object.

Author

Nathan Baker

Returns

Newly allocated object

Definition at line 114 of file [vpParam.c](#).

Here is the call graph for this function:



8.16.2.6 VEXTERNC int Vparam_AtomData_ctor2 (Vparam_AtomData * *thee*)

FORTRAN stub to construct the object.

Author

Nathan Baker

Parameters

<i>thee</i>	Allocated memory
-------------	------------------

Returns

1 if successful, 0 otherwise

Definition at line 126 of file [vpParam.c](#).

Here is the caller graph for this function:

**8.16.2.7 VEXTERNC void Vparam_AtomData_dtor (Vparam_AtomData ** *thee*)**

Destroy object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of object
-------------	--------------------------------------

Definition at line 128 of file [vpParam.c](#).

Here is the call graph for this function:



8.16.2.8 VEXTERNC void Vparam_AtomData_dtor2 (Vparam_AtomData * *thee*)

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object
-------------	-------------------

Definition at line 138 of file [vparam.c](#).

Here is the caller graph for this function:



8.16.2.9 VEXTERNC Vparam* Vparam_ctor ()

Construct the object.

Author

Nathan Baker

Returns

Newly allocated [Vparam](#) object

Definition at line 186 of file [vparam.c](#).

Here is the call graph for this function:

**8.16.2.10 VEXTERNC int Vparam_ctor2 (Vparam * *thee*)**

FORTRAN stub to construct the object.

Author

Nathan Baker

Parameters

<i>thee</i>	Allocated Vparam memory
-------------	---

Returns

1 if successful, 0 otherwise

Definition at line 198 of file [vparam.c](#).

Here is the caller graph for this function:



8.16.2.11 VEXTERNC void Vparam_dtor (Vparam ** *thee*)

Destroy object.

Author

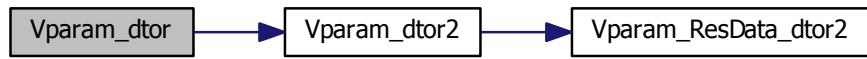
Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of object
-------------	--------------------------------------

Definition at line [218](#) of file [vparam.c](#).

Here is the call graph for this function:



8.16.2.12 VEXTERNC void Vparam_dtor2 (Vparam * *thee*)

FORTRAN stub to destroy object.

Author

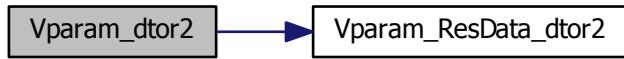
Nathan Baker

Parameters

<i>thee</i>	Pointer to object
-------------	-------------------

Definition at line 228 of file [vparam.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.16.2.13 VEXTERNC Vparam_AtomData* Vparam_getAtomData (Vparam * *thee*, char *resName*[VMAX_ARGLEN], char *atomName*[VMAX_ARGLEN])

Get atom data.

Author

Nathan Baker

Parameters

<i>thee</i>	Vparam object
-------------	---------------

<i>resName</i>	Residue name
<i>atomName</i>	Atom name

Returns

Pointer to the desired atom object or VNULL if residue not found

Note

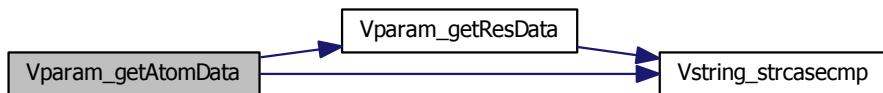
Some method to initialize the database must be called before this method (e.g.,

See also

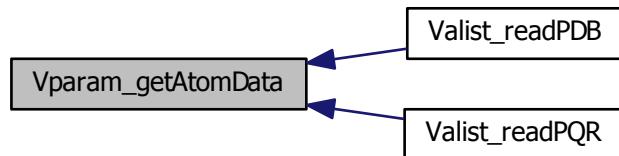
[Vparam_readFlatFile\(\)](#)

Definition at line [272](#) of file [vparam.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.16.2.14 VEXTERNC Vparam_ResData* Vparam_getResData (Vparam * *thee*, char *resName*[VMAX_ARGLEN])

Get residue data.

Author

Nathan Baker

Parameters

<i>thee</i>	Vparam object
<i>resName</i>	Residue name

Returns

Pointer to the desired residue object or VNULL if residue not found

Note

Some method to initialize the database must be called before this method (e.g.,

See also

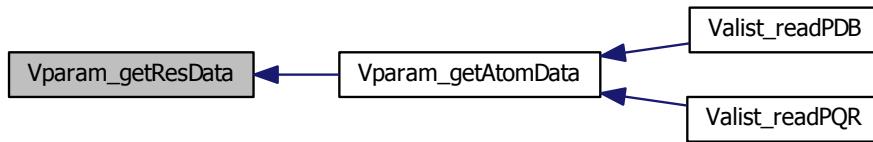
[Vparam_readFlatFile](#))

Definition at line 246 of file [vparam.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.16.2.15 VEXTERNC unsigned long int Vparam_memChk (Vparam * *thee*)

Get number of bytes in this object and its members.

Author

Nathan Baker

Parameters

<i>thee</i>	Vparam object
-------------	---------------

Returns

Number of bytes allocated for object

Definition at line 107 of file [vpParam.c](#).

8.16.2.16 VEXTERNC int Vparam_readFlatFile (Vparam * *thee*, const char * *ioDev*, const char * *ioFmt*, const char * *thost*, const char * *fname*)

Read a flat-file format parameter database.

Author

Nathan Baker

Parameters

<i>thee</i>	Vparam object
<i>ioDev</i>	Input device type (FILE/BUFF/UNIX/INET)

<i>iofmt</i>	Input device format (ASCII/XDR)
<i>thost</i>	Input hostname (for sockets)
<i>fname</i>	Input FILE/BUFF/UNIX/INET name (see note below for format)

Returns

1 if successful, 0 otherwise

Note

The database file should have the following format:

```
RESIDUE ATOM CHARGE RADIUS EPSILON
```

where RESIDUE is the residue name string, ATOM is the atom name string, CHARGE is the charge in e, RADIUS is the van der Waals radius (σ_i) in Å, and EPSILON is the van der Waals well-depth (ε_i) in kJ/mol. See the [Vparam](#) structure documentation for the precise definitions of σ_i and ε_i .

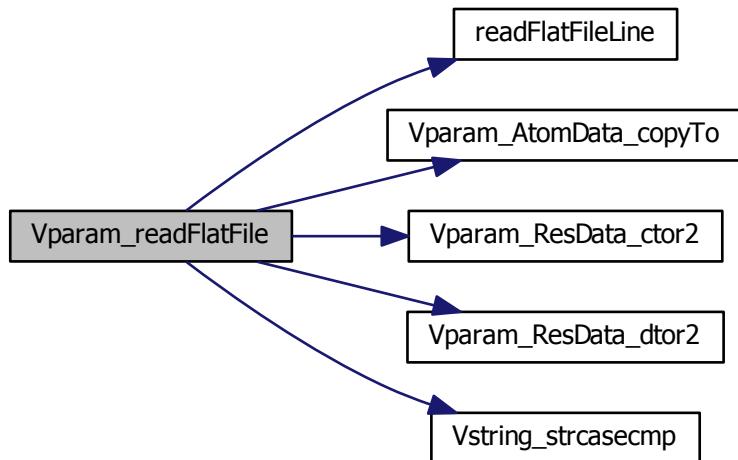
ASCII-format flat files are provided with the APBS source code:

tools/conversion/vparam-amber-par94.dat AMBER parm94 parameters

tools/conversion/vparam-charmm-par_all27.dat CHARMM par_all27_prot_na parameters

Definition at line 450 of file [vparam.c](#).

Here is the call graph for this function:



8.16.2.17 VEXTERNC int Vparam_readXMLFile (Vparam * *thee*, const char * *iodev*, const char * *iofmt*, const char * *thost*, const char * *fname*)

Read an XML format parameter database.

Author

Todd Dolinsky

Parameters

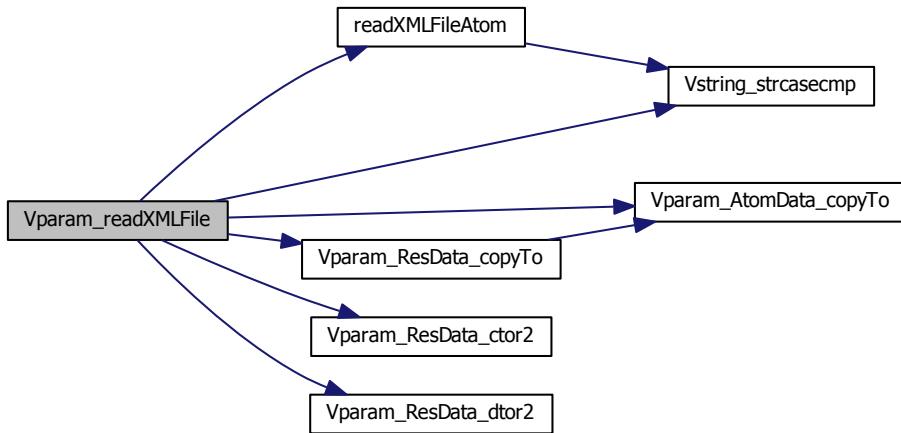
<i>thee</i>	Vparam object
<i>iodev</i>	Input device type (FILE/BUFF/UNIX/INET)
<i>iofmt</i>	Input device format (ASCII/XDR)
<i>thost</i>	Input hostname (for sockets)
<i>fname</i>	Input FILE/BUFF/UNIX/INET name

Returns

1 if successful, 0 otherwise

Definition at line 311 of file [vparam.c](#).

Here is the call graph for this function:



8.16.2.18 VEXTERNC void `Vparam_ResData_copyTo` (`Vparam_ResData * thee,`
`Vparam_ResData * dest`)

Copy current residue object to destination.

Author

Todd Dolinsky

Parameters

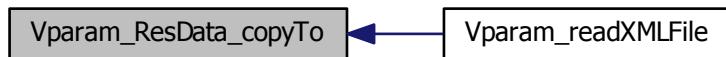
<code>thee</code>	Pointer to source object
<code>dest</code>	Pointer to destination object

Definition at line 590 of file [vparam.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.16.2.19 VEXTERNC Vparam_ResData* Vparam_ResData_ctor(Vmem * mem)

Construct the object.

Author

Nathan Baker

Parameters

<i>mem</i>	Memory object of Vparam master class
------------	--

Returns

Newly allocated object

Definition at line 140 of file [vparam.c](#).

Here is the call graph for this function:



8.16.2.20 VEXTERNC int Vparam_ResData_ctor2 (Vparam_ResData * *thee*, Vmem * *mem*)

FORTRAN stub to construct the object.

Author

Nathan Baker

Parameters

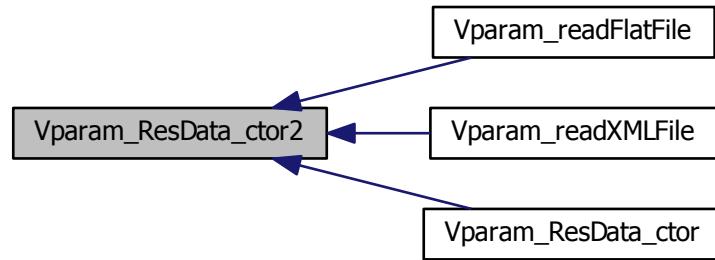
<i>thee</i>	Allocated memory
<i>mem</i>	Memory object of Vparam master class

Returns

1 if successful, 0 otherwise

Definition at line 152 of file [vparam.c](#).

Here is the caller graph for this function:



8.16.2.21 VEXTERN void Vparam_ResData_dtor (Vparam_ResData ** *thee*)

Destroy object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of object
-------------	--------------------------------------

Definition at line 165 of file [vparam.c](#).

Here is the call graph for this function:



8.16.2.22 VEXTERNC void Vparam_ResData_dtor2 (Vparam_ResData * *thee*)

FORTRAN stub to destroy object.

Author

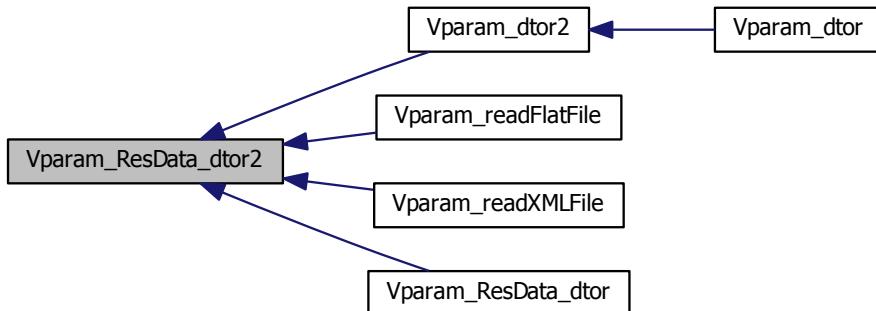
Nathan Baker

Parameters

<i>thee</i>	Pointer to object
-------------	-------------------

Definition at line 175 of file [vparam.c](#).

Here is the caller graph for this function:



8.17 Vpbe class

The Poisson-Boltzmann master class.

Data Structures

- struct **sVpbe**

Contains public data members for Vpbe class/module.

Files

- file `vpbe.h`
Contains declarations for class `Vpbe`.
- file `vpbe.c`
Class `Vpbe` methods.

TypeDefs

- typedef struct `sVpbe Vpbe`
Declaration of the `Vpbe` class as the `Vpbe` structure.

Functions

- VEXTERNC `Valist * Vpbe_getValist (Vpbe *thee)`
Get atom list.
- VEXTERNC `Vacc * Vpbe_getVacc (Vpbe *thee)`
Get accessibility oracle.
- VEXTERNC double `Vpbe_getBulkIonicStrength (Vpbe *thee)`
Get bulk ionic strength.
- VEXTERNC double `Vpbe_getMaxIonRadius (Vpbe *thee)`
Get maximum radius of ion species.
- VEXTERNC double `Vpbe_getTemperature (Vpbe *thee)`
Get temperature.
- VEXTERNC double `Vpbe_getSoluteDiel (Vpbe *thee)`
Get solute dielectric constant.
- VEXTERNC double `Vpbe_getGamma (Vpbe *thee)`
Get apolar coefficient.
- VEXTERNC double `Vpbe_getSoluteRadius (Vpbe *thee)`
Get sphere radius which bounds biomolecule.
- VEXTERNC double `Vpbe_getSoluteXlen (Vpbe *thee)`
Get length of solute in x dimension.
- VEXTERNC double `Vpbe_getSoluteYlen (Vpbe *thee)`
Get length of solute in y dimension.
- VEXTERNC double `Vpbe_getSoluteZlen (Vpbe *thee)`
Get length of solute in z dimension.
- VEXTERNC double * `Vpbe_getSoluteCenter (Vpbe *thee)`
Get coordinates of solute center.
- VEXTERNC double `Vpbe_getSoluteCharge (Vpbe *thee)`

- VEXTERNC double `Vpbe_getSoluteDiel (Vpbe *thee)`
Get total solute charge.
- VEXTERNC double `Vpbe_getSolventDiel (Vpbe *thee)`
Get solvent dielectric constant.
- VEXTERNC double `Vpbe_getSolventRadius (Vpbe *thee)`
Get solvent molecule radius.
- VEXTERNC double `Vpbe_getXkappa (Vpbe *thee)`
Get Debye-Huckel parameter.
- VEXTERNC double `Vpbe_getDeblen (Vpbe *thee)`
Get Debye-Huckel screening length.
- VEXTERNC double `Vpbe_getZkappa2 (Vpbe *thee)`
Get modified squared Debye-Huckel parameter.
- VEXTERNC double `Vpbe_getZmagic (Vpbe *thee)`
Get charge scaling factor.
- VEXTERNC double `Vpbe_getzmem (Vpbe *thee)`
Get z position of the membrane bottom.
- VEXTERNC double `Vpbe_getLmem (Vpbe *thee)`
*Get length of the membrane (A)
author Michael Grabe.*
- VEXTERNC double `Vpbe_getmembraneDiel (Vpbe *thee)`
Get membrane dielectric constant.
- VEXTERNC double `Vpbe_getmemv (Vpbe *thee)`
Get membrane potential (kT)
- VEXTERNC `Vpbe * Vpbe_ctor (Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)`
Construct Vpbe object.
- VEXTERNC int `Vpbe_ctor2 (Vpbe *thee, Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)`
FORTRAN stub to construct Vpbe objct.
- VEXTERNC int `Vpbe_getIons (Vpbe *thee, int *nion, double ionConc[MAXION], double ionRadii[MAXION], double ionQ[MAXION])`
Get information about the counterion species present.
- VEXTERNC void `Vpbe_dtor (Vpbe **thee)`
Object destructor.
- VEXTERNC void `Vpbe_dtor2 (Vpbe *thee)`
FORTRAN stub object destructor.
- VEXTERNC double `Vpbe_getCoulombEnergy1 (Vpbe *thee)`
Calculate coulombic energy of set of charges.
- VEXTERNC unsigned long int `Vpbe_memChk (Vpbe *thee)`
Return the memory used by this structure (and its contents) in bytes.

8.17.1 Detailed Description

The Poisson-Boltzmann master class. Contains objects and parameters used in every PBE calculation, regardless of method.

8.17.2 Function Documentation

8.17.2.1 VEXTERNC *Vpbe *Vpbe_ctor* (Valist * *alist*, int *ionNum*, double * *ionConc*, double * *ionRadii*, double * *ionQ*, double *T*, double *soluteDiel*, double *solventDiel*, double *solventRadius*, int *focusFlag*, double *sdens*, double *z_mem*, double *L*, double *membraneDiel*, double *V*)**

Construct *Vpbe* object.

Author

Nathan Baker and Mike Holst and Michael Grabe

Note

This is partially based on some of Mike Holst's PMG code. Here are a few of the original function comments: kappa is defined as follows:

$$\kappa^2 = \frac{8\pi N_A e_c^2 I_s}{1000 \epsilon_w k_B T}$$

where the units are esu*esu/erg/mol. To obtain cm^{-2} , we multiply by 10^{-16} . Thus, in cm^{-2} , where k_B and e_c are in gaussian rather than mks units, the proper value for kappa is:

$$\kappa^2 = \frac{8\pi N_A e_c^2 I_s}{1000 \epsilon_w k_B T} \times 10^{-16}$$

and the factor of 10^{-16} results from converting cm^2 to angstroms^2 , noting that the 1000 in the denominator has converted m^3 to cm^3 , since the ionic strength I_s is assumed to have been provided in moles per liter, which is moles per 1000 cm^3 .

Returns

Pointer to newly allocated *Vpbe* object

Parameters

<i>alist</i>	Atom list
<i>ionNum</i>	Number of counterion species
<i>ionConc</i>	Array containing counterion concentrations (M)
<i>ionRadii</i>	Array containing counterion radii (A)
<i>ionQ</i>	Array containing counterion charges (e)

<i>T</i>	Temperature for Boltzmann distribution (K)
<i>soluteDiel</i>	Solute internal dielectric constant
<i>solventDiel</i>	Solvent dielectric constant
<i>solventRadius</i>	Solvent probe radius for surfaces that use it (Å)
<i>focusFlag</i>	1 if focusing operation, 0 otherwise
<i>sdens</i>	Vacc sphere density
<i>z_mem</i>	Membrane location (Å)
<i>L</i>	Membrane thickness (Å)
<i>membraneDiel</i>	Membrane dielectric constant
<i>V</i>	Transmembrane potential (V)

Definition at line 247 of file [vpbe.c](#).

```
8.17.2.2 VEXTERNC int Vpbe_ctor2 ( Vpbe * thee, Valist * alist, int ionNum, double *
ionConc, double * ionRadii, double * ionQ, double T, double soluteDiel, double
solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem,
double L, double membraneDiel, double V )
```

FORTRAN stub to construct Vpbe objct.

Author

Nathan Baker and Mike Holst and Michael Grabe

Note

This is partially based on some of Mike Holst's PMG code. Here are a few of the original function comments: kappa is defined as follows:

$$\kappa^2 = \frac{8\pi N_A e_c^2 I_s}{1000 \epsilon \sigma_w k_B T}$$

where the units are esu*esu/erg/mol. To obtain cm^{-2} , we multiply by 10^{-16} . Thus, in cm^{-2} , where k_B and e_c are in gaussian rather than mks units, the proper value for kappa is:

$$\kappa^2 = \frac{8\pi N_A e_c^2 I_s}{1000 \epsilon \sigma_w k_B T} \times 10^{-16}$$

and the factor of 10^{-16} results from converting cm^{-2} to angstroms^{-2} , noting that the 1000 in the denominator has converted m^{-3} to cm^{-3} , since the ionic strength I_s is assumed to have been provided in moles per liter, which is moles per 1000 cm^{-3} .

Bug

The focusing flag is currently not used!!

Returns

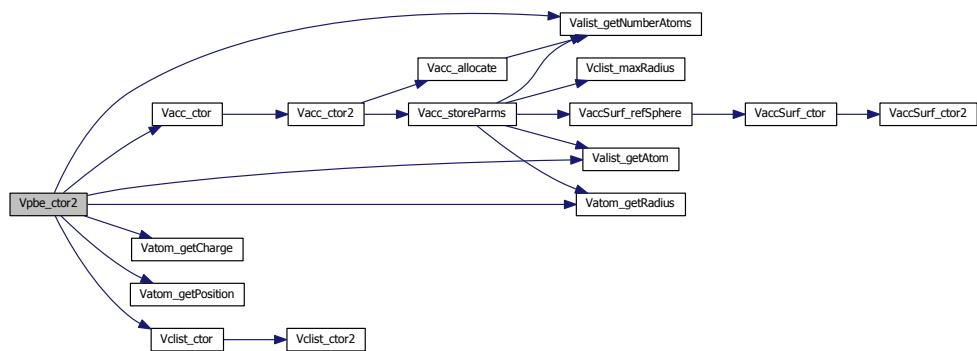
1 if successful, 0 otherwise

Parameters

<i>thee</i>	Pointer to memory allocated for Vpbe object
<i>alist</i>	Atom list
<i>ionNum</i>	Number of counterion species
<i>ionConc</i>	Array containing counterion concentrations (M)
<i>ionRadii</i>	Array containing counterion radii (A)
<i>ionQ</i>	Array containing counterion charges (e)
<i>T</i>	Temperature for Boltzmann distribution (K)
<i>soluteDiel</i>	Solute internal dielectric constant
<i>solventDiel</i>	Solvent dielectric constant
<i>solventRadius</i>	Solvent probe radius for surfaces that use it (A)
<i>focusFlag</i>	1 if focusing operation, 0 otherwise
<i>sdens</i>	Vacc sphere density
<i>z_mem</i>	Membrane location (A)
<i>L</i>	Membrane thickness (A)
<i>membraneDiel</i>	Membrane dielectric constant
<i>V</i>	Transmembrane potential (V)

Definition at line 265 of file [vpbe.c](#).

Here is the call graph for this function:



8.17.2.3 VEXTERNC void Vpbe_dtor (*Vpbe* ** *thee*)

Object destructor.

Author

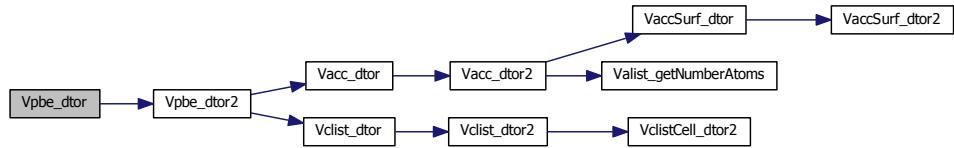
Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of object to be destroyed
-------------	--

Definition at line 468 of file [vpbe.c](#).

Here is the call graph for this function:



8.17.2.4 VEXTERNC void Vpbe_dtor2 (*Vpbe* * *thee*)

FORTRAN stub object destructor.

Author

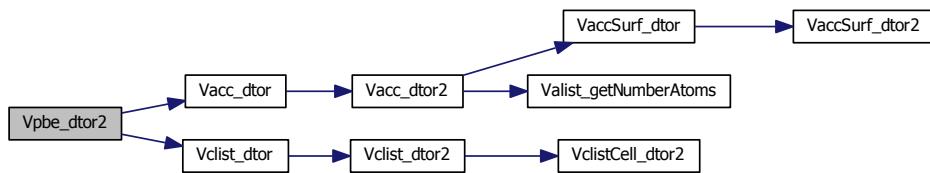
Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
-------------	-----------------------------------

Definition at line 476 of file [vpbe.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.2.5 VEXTERNC double Vpbe_getBulkIonicStrength (Vpbe * *thee*)

Get bulk ionic strength.

Author

Nathan Baker

Parameters

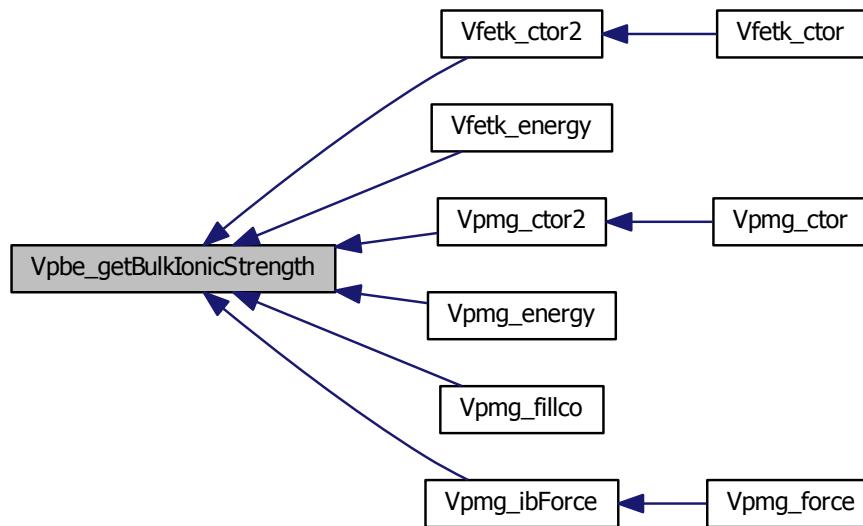
<i>thee</i>	Vpbe object
-------------	-------------

Returns

Bulk ionic strength (M)

Definition at line 85 of file [vpbe.c](#).

Here is the caller graph for this function:



8.17.2.6 VEXTERNC double Vpbe_getCoulombEnergy1 (Vpbe * *thee*)

Calculate coulombic energy of set of charges.

Perform an inefficient double sum to calculate the Coulombic energy of a set of charges in a homogeneous dielectric (with permittivity equal to the protein interior) and zero ionic strength. Result is returned in units of $k_B T$. The sum can be restriction to charges present in simplices of specified color (pcolor); if (color == -1) no restrictions are used.

Author

Nathan Baker

Parameters

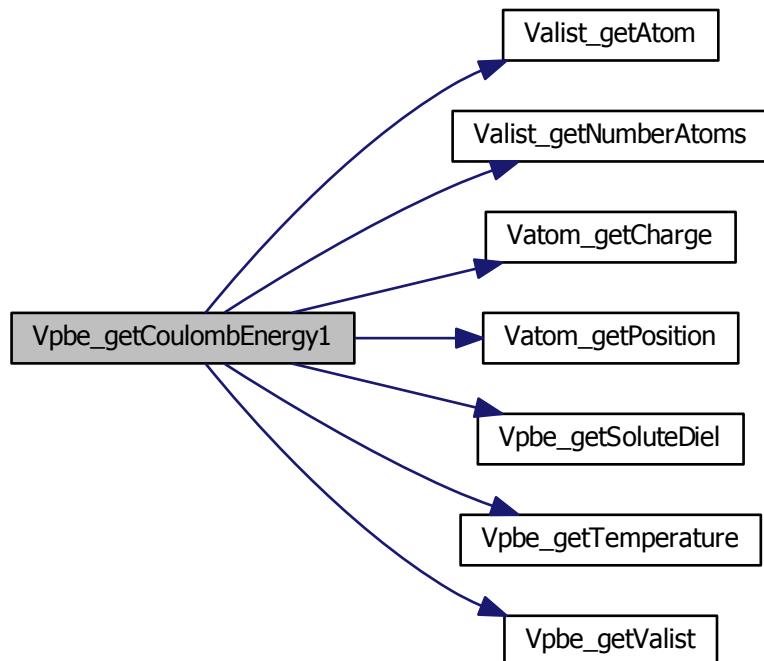
<i>thee</i>	Vpbe object
-------------	-------------

Returns

Coulombic energy in units of $k_B T$.

Definition at line 482 of file [vpbe.c](#).

Here is the call graph for this function:



8.17.2.7 VEXTERNC double `Vpbe_getDeblen (Vpbe * thee)`

Get Debye-Hückel screening length.

Author

Nathan Baker

Parameters

<code>thee</code>	Vpbe object
-------------------	-------------

Returns

Debye-Hückel screening length (Å)

Definition at line 142 of file [vpbe.c](#).

8.17.2.8 VEXTERNC double Vpbe_getGamma (Vpbe * *thee*)

Get apolar coefficient.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Apolar coefficient (kJ/mol/A^2)

8.17.2.9 VEXTERNC int Vpbe_getIons (Vpbe * *thee*, int * *nion*, double *ionConc*[MAXION], double *ionRadii*[MAXION], double *ionQ*[MAXION])

Get information about the counterion species present.

Author

Nathan Baker

Parameters

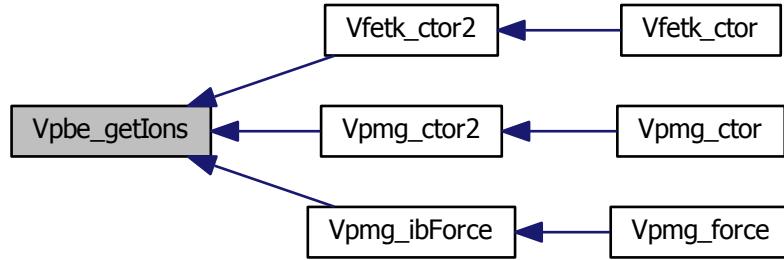
<i>thee</i>	Pointer to Vpbe object
<i>nion</i>	Set to the number of counterion species
<i>ionConc</i>	Array to store counterion species' concentrations (M)
<i>ionRadii</i>	Array to store counterion species' radii (Å)
<i>ionQ</i>	Array to store counterion species' charges (e)

Returns

Number of ions

Definition at line 536 of file [vpbe.c](#).

Here is the caller graph for this function:



8.17.2.10 VEXTERNC double Vpbe_getLmem (`Vpbe * thee`)

Get length of the membrane (A)

aauthor Michael Grabe.

Parameters

<code>thee</code>	<code>Vpbe object</code>
-------------------	--------------------------

Returns

Length of the membrane (A)

Definition at line 210 of file [vpbe.c](#).

8.17.2.11 VEXTERNC double Vpbe_getMaxIonRadius (`Vpbe * thee`)

Get maximum radius of ion species.

Author

Nathan Baker

Parameters

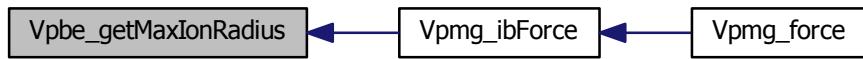
<code>thee</code>	<code>Vpbe object</code>
-------------------	--------------------------

Returns

Maximum radius (A)

Definition at line 128 of file [vpbe.c](#).

Here is the caller graph for this function:

**8.17.2.12 VEXTERNC double Vpbe_getmembraneDiel (*Vpbe * thee*)**

Get membrane dielectric constant.

Author

Michael Grabe

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Membrane dielectric constant

Definition at line 222 of file [vpbe.c](#).

8.17.2.13 VEXTERNC double Vpbe_getmemv (*Vpbe * thee*)

Get membrane potential (kT)

Author

Michael Grabe

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Definition at line [234](#) of file [vpbe.c](#).

8.17.2.14 VEXTERNC double* Vpbe_getSoluteCenter (Vpbe * *thee*)

Get coordinates of solute center.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Pointer to 3*double array with solute center coordinates (A)

Definition at line [108](#) of file [vpbe.c](#).

8.17.2.15 VEXTERNC double Vpbe_getSoluteCharge (Vpbe * *thee*)

Get total solute charge.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Total solute charge (e)

Definition at line [187](#) of file [vpbe.c](#).

8.17.2.16 VEXTERNC double Vpbe_getSoluteDiel (Vpbe * *thee*)

Get solute dielectric constant.

Author

Nathan Baker

Parameters

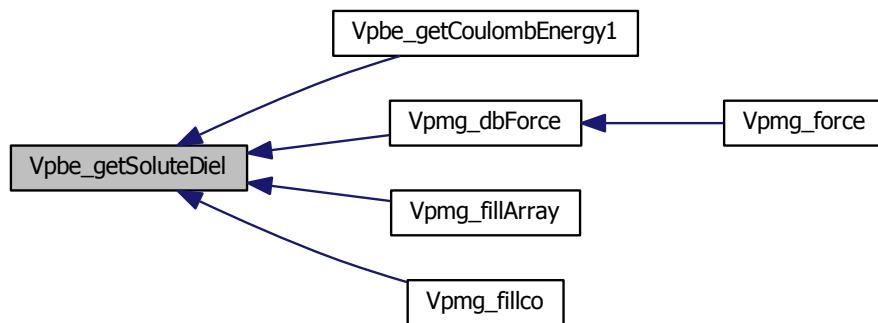
<i>thee</i>	Vpbe object
-------------	-------------

Returns

Solute dielectric constant

Definition at line 100 of file [vpbe.c](#).

Here is the caller graph for this function:



8.17.2.17 VEXTERNC double Vpbe_getSoluteRadius (`Vpbe * thee`)

Get sphere radius which bounds biomolecule.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Sphere radius which bounds biomolecule (A)

Definition at line 163 of file [vpbe.c](#).

8.17.2.18 VEXTERNC double Vpbe_getSoluteXlen (Vpbe * *thee*)

Get length of solute in x dimension.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Length of solute in x dimension (A)

Definition at line [169](#) of file [vpbe.c](#).

8.17.2.19 VEXTERNC double Vpbe_getSoluteYlen (Vpbe * *thee*)

Get length of solute in y dimension.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Length of solute in y dimension (A)

Definition at line [175](#) of file [vpbe.c](#).

8.17.2.20 VEXTERNC double Vpbe_getSoluteZlen (Vpbe * *thee*)

Get length of solute in z dimension.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Length of solute in z dimension (A)

Definition at line 181 of file [vpbe.c](#).

8.17.2.21 VEXTERNC double Vpbe_getSolventDiel (Vpbe * *thee*)

Get solvent dielectric constant.

Author

Nathan Baker

Parameters

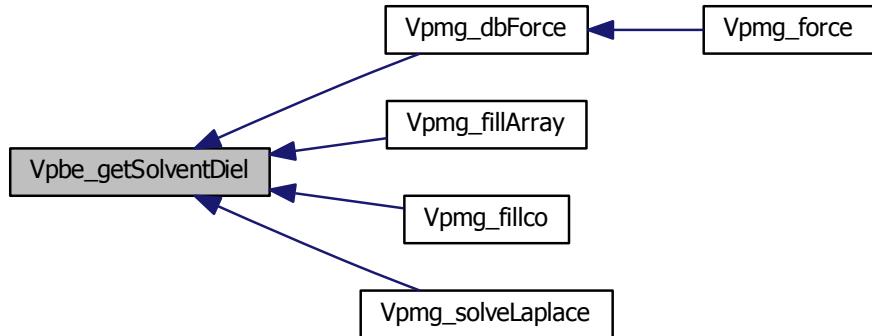
<i>thee</i>	Vpbe object
-------------	-------------

Returns

Solvent dielectric constant

Definition at line 114 of file [vpbe.c](#).

Here is the caller graph for this function:



8.17.2.22 VEXTERNC double Vpbe_getSolventRadius (Vpbe * *thee*)

Get solvent molecule radius.

Author

Nathan Baker

Parameters

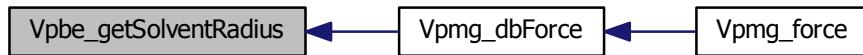
<i>thee</i>	Vpbe object
-------------	-------------

Returns

Solvent molecule radius (A)

Definition at line 121 of file [vpbe.c](#).

Here is the caller graph for this function:

**8.17.2.23 VEXTERNC double Vpbe_getTemperature (Vpbe * *thee*)**

Get temperature.

Author

Nathan Baker

Parameters

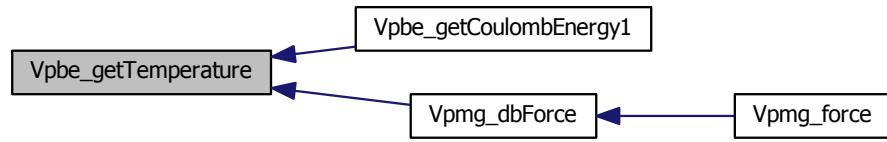
<i>thee</i>	Vpbe object
-------------	-------------

Returns

Temperature (K)

Definition at line 92 of file [vpbe.c](#).

Here is the caller graph for this function:



8.17.2.24 VEXTERNC `Vacc*` `Vpbe_getVacc(Vpbe *thee)`

Get accessibility oracle.

Author

Nathan Baker

Parameters

<code>thee</code>	<code>Vpbe object</code>
-------------------	--------------------------

Returns

Pointer to internal `Vacc` object

Definition at line 77 of file [vpbe.c](#).

Here is the caller graph for this function:



8.17.2.25 VEXTERNC Valist* Vpbe_getValist (Vpbe * *thee*)

Get atom list.

Author

Nathan Baker

Parameters

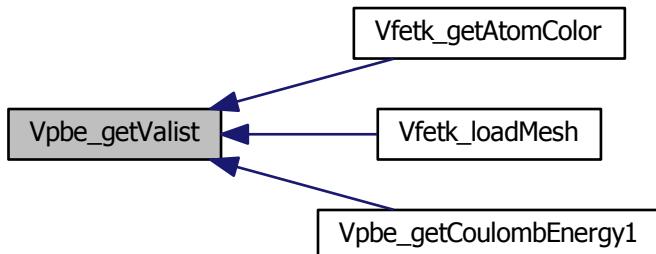
<i>thee</i>	Vpbe object
-------------	-------------

Returns

Pointer to internal Valist object

Definition at line [70](#) of file [vpbe.c](#).

Here is the caller graph for this function:

**8.17.2.26 VEXTERNC double Vpbe_getXkappa (Vpbe * *thee*)**

Get Debye-Huckel parameter.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Bulk Debye-Hückel parameter (Å)

Definition at line 135 of file [vpbe.c](#).

8.17.2.27 VEXTERNC double Vpbe_getZkappa2 (Vpbe * *thee*)

Get modified squared Debye-Hückel parameter.

Author

Nathan Baker

Parameters

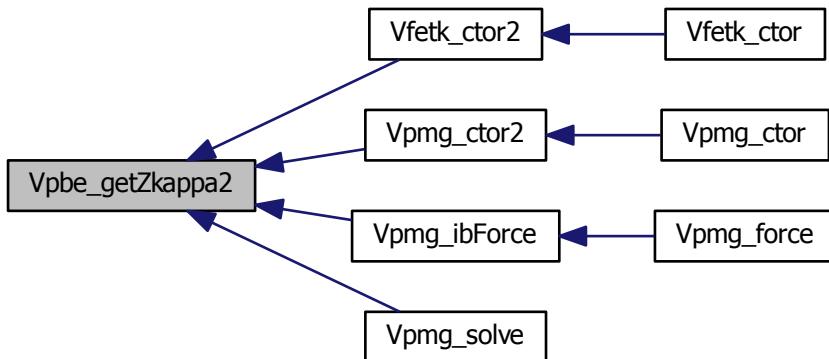
<i>thee</i>	Vpbe object
-------------	-------------

Returns

Modified squared Debye-Hückel parameter (Å^{-2})

Definition at line 149 of file [vpbe.c](#).

Here is the caller graph for this function:



8.17.2.28 VEXTERNC double Vpbe_getZmagic (Vpbe * *thee*)

Get charge scaling factor.

Author

Nathan Baker and Mike Holst

Parameters

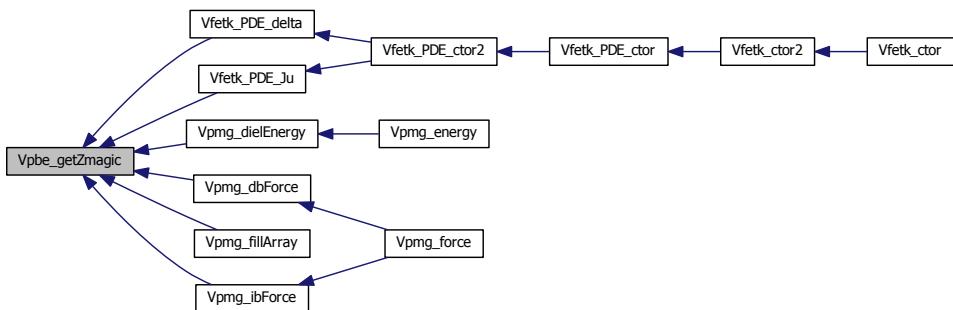
<i>thee</i>	Vpbe object
-------------	-------------

Returns

Get factor for scaling charges (in e) to internal units

Definition at line 156 of file [vpbe.c](#).

Here is the caller graph for this function:



8.17.2.29 VEXTERNC double Vpbe_getzmem (Vpbe * *thee*)

Get z position of the membrane bottom.

Author

Michael Grabe

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

z value of membrane (A)

Definition at line 198 of file [vpbe.c](#).

8.17.2.30 VEXTERNC unsigned long int Vpbe_memChk (Vpbe * *thee*)

Return the memory used by this structure (and its contents) in bytes.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

The memory used by this structure and its contents in bytes

Definition at line 524 of file [vpbe.c](#).

Here is the call graph for this function:



8.18 Vstring class

Provides a collection of useful non-ANSI string functions.

Files

- file [vstring.h](#)

Contains declarations for class Vstring.

- file [vstring.c](#)

Class Vstring methods.

Functions

- VEXTERNC int [Vstring_strcasecmp](#) (const char *s1, const char *s2)
Case-insensitive string comparison (BSD standard)
- VEXTERNC int [Vstring_isdigit](#) (const char *tok)
A modified sscanf that examines the complete string.

8.18.1 Detailed Description

Provides a collection of useful non-ANSI string functions.

8.18.2 Function Documentation

8.18.2.1 VEXTERNC int Vstring_isdigit (const char * tok)

A modified sscanf that examines the complete string.

Author

Todd Dolinsky

Parameters

<i>tok</i>	The string to examine
------------	-----------------------

Returns

1 if the entire string is an integer, 0 if otherwise.

Definition at line 130 of file [vstring.c](#).

8.18.2.2 VEXTERNC int Vstring_strcasecmp (const char * s1, const char * s2)

Case-insensitive string comparison (BSD standard)

Author

Copyright (c) 1988-1993 The Regents of the University of California. Copyright (c) 1995-1996 Sun Microsystems, Inc.

Note

Copyright (c) 1988-1993 The Regents of the University of California. Copyright (c) 1995-1996 Sun Microsystems, Inc.

Parameters

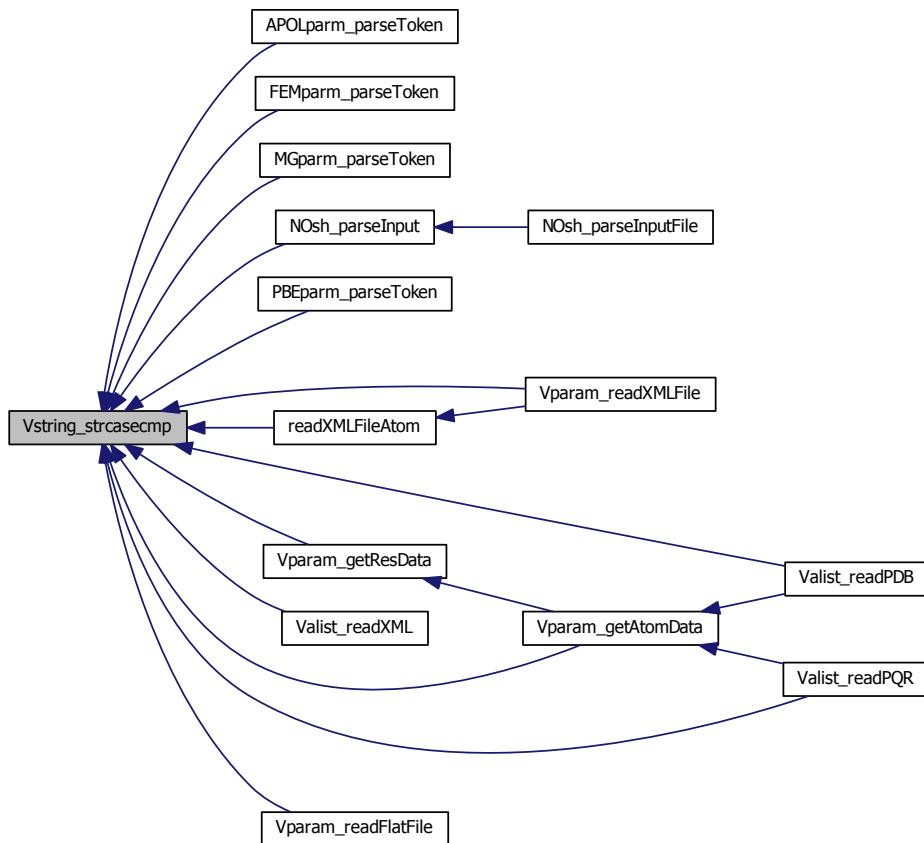
<i>s1</i>	First string for comparison
<i>s2</i>	Second string for comparison

Returns

An integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2. (Source: Linux man pages)

Definition at line 66 of file [vstring.c](#).

Here is the caller graph for this function:



8.19 Vunit class

Collection of constants and conversion factors.

Files

- file [vunit.h](#)

Contains a collection of useful constants and conversion factors.

Defines

- #define `Vunit_J_to_cal` 4.1840000e+00
Multiply by this to convert J to cal.
- #define `Vunit_cal_to_J` 2.3900574e-01
Multiply by this to convert cal to J.
- #define `Vunit_amu_to_kg` 1.6605402e-27
Multiply by this to convert amu to kg.
- #define `Vunit_kg_to_amu` 6.0221367e+26
Multiply by this to convert kg to amu.
- #define `Vunit_ec_to_C` 1.6021773e-19
Multiply by this to convert ec to C.
- #define `Vunit_C_to_ec` 6.2415065e+18
Multiply by this to convert C to ec.
- #define `Vunit_ec` 1.6021773e-19
Charge of an electron in C.
- #define `Vunit_kb` 1.3806581e-23
Boltzmann constant.
- #define `Vunit_Na` 6.0221367e+23
Avogadro's number.
- #define `Vunit_pi` VPI
Pi.
- #define `Vunit_eps0` 8.8541878e-12
Vacuum permittivity.
- #define `Vunit_esu_ec2A` 3.3206364e+02
 $e_c^2 / \text{in ESU units} \Rightarrow \text{kcal/mol}$
- #define `Vunit_esu_kb` 1.9871913e-03
 $k_b \text{ in ESU units} \Rightarrow \text{kcal/mol}$

8.19.1 Detailed Description

Collection of constants and conversion factors.

8.20 Vgrid class

Oracle for Cartesian mesh data.

Data Structures

- struct [sVgrid](#)
Electrostatic potential oracle for Cartesian mesh data.

Files

- file [vgrid.h](#)
Potential oracle for Cartesian mesh data.
- file [vgrid.c](#)
Class Vgrid methods.

Defines

- #define [VGRID_DIGITS](#) 6
Number of decimal places for comparisons and formatting.

Typedefs

- typedef struct [sVgrid](#) Vgrid
Declaration of the Vgrid class as the [sVgrid](#) structure.

Functions

- VEXTERNC unsigned long int [Vgrid_memChk](#) ([Vgrid](#) *thee)
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC [Vgrid](#) * [Vgrid_ctor](#) (int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)
Construct Vgrid object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int [Vgrid_ctor2](#) ([Vgrid](#) *thee, int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)
Initialize Vgrid object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int [Vgrid_value](#) ([Vgrid](#) *thee, double x[3], double *value)
Get potential value (from mesh or approximation) at a point.
- VEXTERNC void [Vgrid_dtor](#) ([Vgrid](#) **thee)
Object destructor.
- VEXTERNC void [Vgrid_dtor2](#) ([Vgrid](#) *thee)
FORTRAN stub object destructor.

- VEXTERNC int `Vgrid_curvature` (`Vgrid *thee`, double `pt[3]`, int `cflag`, double `*curv`)

Get second derivative values at a point.

- VEXTERNC int `Vgrid_gradient` (`Vgrid *thee`, double `pt[3]`, double `grad[3]`)

Get first derivative values at a point.

- VEXTERNC int `Vgrid_readGZ` (`Vgrid *thee`, const char `*fname`)

Read in OpenDX data in GZIP format.

- VEXTERNC void `Vgrid_writeUHBD` (`Vgrid *thee`, const char `*iodev`, const char `*iofmt`, const char `*thost`, const char `*fname`, char `*title`, double `*pvec`)

Write out the data in UHBD grid format.

- VEXTERNC void `Vgrid_writeDX` (`Vgrid *thee`, const char `*iodev`, const char `*iofmt`, const char `*thost`, const char `*fname`, char `*title`, double `*pvec`)

Write out the data in OpenDX grid format.

- VEXTERNC int `Vgrid_readDX` (`Vgrid *thee`, const char `*iodev`, const char `*iofmt`, const char `*thost`, const char `*fname`)

Read in data in OpenDX grid format.

- VEXTERNC double `Vgrid_integrate` (`Vgrid *thee`)

Get the integral of the data.

- VEXTERNC double `Vgrid_normL1` (`Vgrid *thee`)

Get the L_1 norm of the data. This returns the integral:

$$\|u\|_{L_1} = \int_{\Omega} |u(x)| dx$$

- VEXTERNC double `Vgrid_normL2` (`Vgrid *thee`)

Get the L_2 norm of the data. This returns the integral:

$$\|u\|_{L_2} = \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

- VEXTERNC double `Vgrid_normLinf` (`Vgrid *thee`)

Get the L_∞ norm of the data. This returns the integral:

$$\|u\|_{L_\infty} = \sup_{x \in \Omega} |u(x)|$$

- VEXTERNC double `Vgrid_seminormH1` (`Vgrid *thee`)

Get the H_1 semi-norm of the data. This returns the integral:

$$|u|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx \right)^{1/2}$$

- VEXTERNC double `Vgrid_normH1` (`Vgrid *thee`)

Get the H_1 norm (or energy norm) of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx + \int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

8.20.1 Detailed Description

Oracle for Cartesian mesh data.

8.20.2 Function Documentation

8.20.2.1 **VEXTERNC Vgrid* Vgrid_ctor (int *nx*, int *ny*, int *nz*, double *hx*, double *hy*, double *hzed*, double *xmin*, double *ymin*, double *zmin*, double * *data*)**

Construct Vgrid object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

<i>nx</i>	Number grid points in x direction
<i>ny</i>	Number grid points in y direction
<i>nz</i>	Number grid points in z direction
<i>hx</i>	Grid spacing in x direction
<i>hy</i>	Grid spacing in y direction
<i>hzed</i>	Grid spacing in z direction
<i>xmin</i>	x coordinate of lower grid corner
<i>ymin</i>	y coordinate of lower grid corner
<i>zmin</i>	z coordinate of lower grid corner
<i>data</i>	<i>nx</i> * <i>ny</i> * <i>nz</i> array of data. This can be VNULL if you are planning to read in data later with one of the read routines

Returns

Newly allocated and initialized Vgrid object

Definition at line 78 of file [vgrid.c](#).

Here is the caller graph for this function:



8.20.2.2 VEXTERNC int Vgrid_ctor2 (*Vgrid *thee, int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data*)

Initialize Vgrid object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to newly allocated Vgrid object
<i>nx</i>	Number grid points in x direction
<i>ny</i>	Number grid points in y direction
<i>nz</i>	Number grid points in z direction
<i>hx</i>	Grid spacing in x direction
<i>hy</i>	Grid spacing in y direction
<i>hzed</i>	Grid spacing in z direction
<i>xmin</i>	x coordinate of lower grid corner
<i>ymin</i>	y coordinate of lower grid corner
<i>zmin</i>	z coordinate of lower grid corner
<i>data</i>	nx*ny*nz array of data. This can be VNULL if you are planning to read in data later with one of the read routines

Returns

Newly allocated and initialized Vgrid object

Definition at line 97 of file [vgrid.c](#).

8.20.2.3 VEXTERNC int Vgrid_curvature (*Vgrid *thee, double pt[3], int cflag, double *curv*)

Get second derivative values at a point.

Author

Steve Bond and Nathan Baker

Parameters

<i>thee</i>	Pointer to Vgrid object
<i>pt</i>	Location to evaluate second derivative
<i>cflag</i>	<ul style="list-style-type: none"> • 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature

<i>curv</i>	Specified curvature value
-------------	---------------------------

Returns

1 if successful, 0 if off grid

Definition at line [282](#) of file [vgrid.c](#).

Here is the caller graph for this function:

**8.20.2.4 VEXTERNC void Vgrid_dtor (Vgrid ** *thee*)**

Object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of object to be destroyed
-------------	--

Definition at line [137](#) of file [vgrid.c](#).

Here is the caller graph for this function:



8.20.2.5 VEXTERNC void Vgrid_dtor2 (Vgrid * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
-------------	-----------------------------------

Definition at line 150 of file [vgrid.c](#).

8.20.2.6 VEXTERNC int Vgrid_gradient (Vgrid * *thee*, double *pt*[3], double *grad*[3])

Get first derivative values at a point.

Author

Nathan Baker and Steve Bond

Parameters

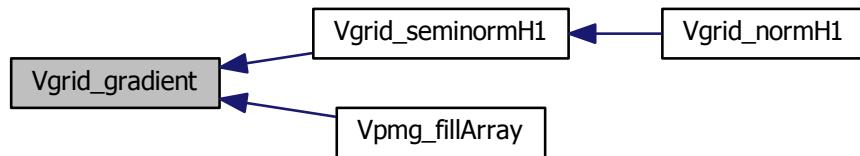
<i>thee</i>	Pointer to Vgrid object
<i>pt</i>	Location to evaluate gradient
<i>grad</i>	Gradient

Returns

1 if successful, 0 if off grid

Definition at line 362 of file [vgrid.c](#).

Here is the caller graph for this function:



8.20.2.7 VEXTERNC double Vgrid_integrate (`Vgrid * thee`)

Get the integral of the data.

Author

Nathan Baker

Parameters

<code>thee</code>	Vgrid object
-------------------	--------------

Returns

Integral of data

Definition at line 1348 of file [vgrid.c](#).

8.20.2.8 VEXTERNC unsigned long int Vgrid_memChk (`Vgrid * thee`)

Return the memory used by this structure (and its contents) in bytes.

Author

Nathan Baker

Parameters

<code>thee</code>	Vgrid object
-------------------	--------------

Returns

The memory used by this structure and its contents in bytes

Definition at line 63 of file [vgrid.c](#).

8.20.2.9 VEXTERNC double Vgrid_normH1 (Vgrid * *thee*)

Get the H_1 norm (or energy norm) of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx + \int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

Author

Nathan Baker

Parameters

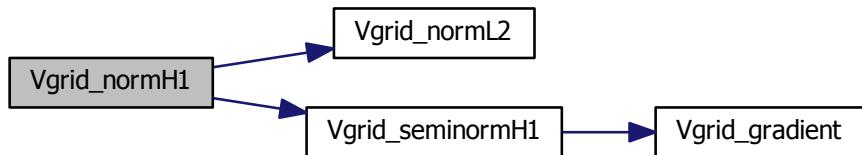
<i>thee</i>	Vgrid object
-------------	--------------

Returns

Integral of data

Definition at line 1485 of file [vgrid.c](#).

Here is the call graph for this function:



8.20.2.10 VEXTERNC double Vgrid_normL1 (Vgrid * *thee*)

Get the L_1 norm of the data. This returns the integral:

$$\|u\|_{L_1} = \int_{\Omega} |u(x)| dx$$

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

L_1 norm of data

Definition at line 1385 of file [vgrid.c](#).

8.20.2.11 VEXTERNC double Vgrid_normL2 (Vgrid * *thee*)

Get the L_2 norm of the data. This returns the integral:

$$\|u\|_{L_2} = \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

L_2 norm of data

Definition at line 1414 of file [vgrid.c](#).

Here is the caller graph for this function:



8.20.2.12 VEXTERNC double Vgrid_normLinf (Vgrid * *thee*)

Get the L_∞ norm of the data. This returns the integral:

$$\|u\|_{L_\infty} = \sup_{x \in \Omega} |u(x)|$$

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

L_∞ norm of data

Definition at line 1500 of file [vgrid.c](#).

8.20.2.13 VEXTERNC int Vgrid_readDX (Vgrid * *thee*, const char * *iodev*, const char * *iofmt*, const char * *thost*, const char * *fname*)

Read in data in OpenDX grid format.

Note

All dimension information is given in order: z, y, x

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
<i>iodev</i>	Input device type (FILE/BUFF/UNIX/INET)
<i>iofmt</i>	Input device format (ASCII/XDR)
<i>thost</i>	Input hostname (for sockets)
<i>fname</i>	Input FILE/BUFF/UNIX/INET name

Returns

1 if sucessful, 0 otherwise

Definition at line 583 of file [vgrid.c](#).

8.20.2.14 VEXTERNC int Vgrid_readGZ (Vgrid * *thee*, const char * *fname*)

Read in OpenDX data in GZIP format.

Author

Dave Gohara

Returns

1 if successful, 0 otherwise

Parameters

<i>thee</i>	Object with grid data to write
<i>fname</i>	Path to write to

Definition at line 445 of file [vgrid.c](#).

8.20.2.15 VEXTERNC double Vgrid_seminormH1 (Vgrid * *thee*)

Get the H_1 semi-norm of the data. This returns the integral:

$$|u|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx \right)^{1/2}$$

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

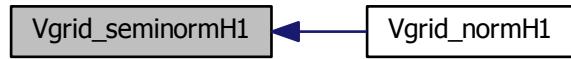
Integral of data

Definition at line 1443 of file [vgrid.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**8.20.2.16 VEXTERNC int Vgrid_value (Vgrid * *thee*, double *x*[3], double * *value*)**

Get potential value (from mesh or approximation) at a point.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid obejct
<i>x</i>	Point at which to evaluate potential
<i>value</i>	Value of data at point x

Returns

1 if successful, 0 if off grid

Definition at line 164 of file [vgrid.c](#).

Here is the caller graph for this function:



8.20.2.17 VEXTERNC void Vgrid_writeDX (*Vgrid * thee, const char * iodev, const char * ifofmt, const char * thost, const char * fname, char * title, double * pvec*)

Write out the data in OpenDX grid format.

Author

Nathan Baker

Parameters

<i>thee</i>	Grid object
<i>iodev</i>	Output device type (FILE/BUFF/UNIX/INET)
<i>iofmt</i>	Output device format (ASCII/XDR)
<i>thost</i>	Output hostname (for sockets)
<i>fname</i>	Output FILE/BUFF/UNIX/INET name
<i>title</i>	Title to be inserted in grid file
<i>pvec</i>	Partition weight (if 1: point in current partition, if 0 point not in current partition if > 0 && < 1 point on/near boundary)

Definition at line 1001 of file [vgrid.c](#).

8.20.2.18 VEXTERNC void Vgrid_writeUHBD (*Vgrid * thee, const char * iodev, const char * ifofmt, const char * thost, const char * fname, char * title, double * pvec*)

Write out the data in UHBD grid format.

Note

- The mesh spacing should be uniform
- Format changed from %12.6E to %12.5E

Author

Nathan Baker

Parameters

<i>thee</i>	Grid object
<i>iodev</i>	Output device type (FILE/BUFF/UNIX/INET)
<i>iofmt</i>	Output device format (ASCII/XDR)
<i>thost</i>	Output hostname (for sockets)
<i>fname</i>	Output FILE/BUFF/UNIX/INET name
<i>title</i>	Title to be inserted in grid file
<i>pvec</i>	Partition weight (if 1: point in current partition, if 0 point not in current partition if > 0 && < 1 point on/near boundary)

Bug

This routine does not respect partition information

Definition at line 1251 of file [vgrid.c](#).

8.21 Vmgrid class

Oracle for Cartesian mesh data.

Data Structures

- struct [sVmgrid](#)

Multiresolution oracle for Cartesian mesh data.

Files

- file [vmgrid.h](#)

Multiresolution oracle for Cartesian mesh data.

- file [vmgrid.c](#)

Class Vmgrid methods.

Defines

- #define **VMGRIDMAX** 20
The maximum number of levels in the grid hierarchy.

Typedefs

- typedef struct **sVmgrid** **Vmgrid**
Declaration of the Vmgrid class as the Vgmrid structure.

Functions

- VEXTERNC **Vmgrid** * **Vmgrid_ctor** ()
Construct Vmgrid object.
- VEXTERNC int **Vmgrid_ctor2** (**Vmgrid** *thee)
Initialize Vmgrid object.
- VEXTERNC int **Vmgrid_value** (**Vmgrid** *thee, double x[3], double *value)
Get potential value (from mesh or approximation) at a point.
- VEXTERNC void **Vmgrid_dtor** (**Vmgrid** **thee)
Object destructor.
- VEXTERNC void **Vmgrid_dtor2** (**Vmgrid** *thee)
FORTRAN stub object destructor.
- VEXTERNC int **Vmgrid_addGrid** (**Vmgrid** *thee, **Vgrid** *grid)
Add a grid to the hierarchy.
- VEXTERNC int **Vmgrid_curvature** (**Vmgrid** *thee, double pt[3], int cflag, double *curv)
Get second derivative values at a point.
- VEXTERNC int **Vmgrid_gradient** (**Vmgrid** *thee, double pt[3], double grad[3])
Get first derivative values at a point.
- VEXTERNC **Vgrid** * **Vmgrid_getGridByNum** (**Vmgrid** *thee, int num)
Get specific grid in hierarchy.
- VEXTERNC **Vgrid** * **Vmgrid_getGridByPoint** (**Vmgrid** *thee, double pt[3])
Get grid in hierarchy which contains specified point or VNULL.

8.21.1 Detailed Description

Oracle for Cartesian mesh data.

8.21.2 Function Documentation

8.21.2.1 VEXTERNC int Vmgrid_addGrid (*Vmgrid * thee, Vgrid * grid*)

Add a grid to the hierarchy.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
<i>grid</i>	Grid to be added. As mentioned above, we would prefer to have the finest grid added first, next-finest second, ..., coarsest last -- this is how the grid will be searched when looking up values for points. However, this is not enforced to provide flexibility for cases where the dataset is decomposed into disjoint partitions, etc.

Returns

1 if successful, 0 otherwise

Definition at line 204 of file [vmgrid.c](#).

8.21.2.2 VEXTERNC *Vmgrid** Vmgrid_ctor ()

Construct Vmgrid object.

Author

Nathan Baker

Returns

Newly allocated and initialized Vmgrid object

Definition at line 66 of file [vmgrid.c](#).

8.21.2.3 VEXTERNC int Vmgrid_ctor2 (*Vmgrid * thee*)

Initialize Vmgrid object.

Author

Nathan Baker

Parameters

<i>thee</i>	Newly allocated Vmgrid object
-------------	-------------------------------

Returns

Newly allocated and initialized Vmgrid object

Definition at line 81 of file [vmgrid.c](#).

8.21.2.4 VEXTERNC int Vmgrid_curvature (Vmgrid * *thee*, double *pt*[3], int *cflag*, double * *curv*)

Get second derivative values at a point.

Author

Nathan Baker (wrapper for Vgrid routine by Steve Bond)

Parameters

<i>thee</i>	Pointer to Vmgrid object
<i>pt</i>	Location to evaluate second derivative
<i>cflag</i>	<ul style="list-style-type: none">• 0: Reduced Maximal Curvature• 1: Mean Curvature (Laplace)• 2: Gauss Curvature• 3: True Maximal Curvature
<i>curv</i>	Specified curvature value

Returns

1 if successful, 0 if off grid

Definition at line 147 of file [vmgrid.c](#).

8.21.2.5 VEXTERNC void Vmgrid_dtor (Vmgrid ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of object to be destroyed
-------------	--

Definition at line 97 of file [vmgrid.c](#).

8.21.2.6 VEXTERNC void Vmgrid_dtor2(Vmgrid * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
-------------	-----------------------------------

Definition at line 110 of file [vmgrid.c](#).

8.21.2.7 VEXTERNC Vgrid* Vmgrid_getGridByNum(Vmgrid * *thee*, int *num*)

Get specific grid in hierarchy.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vmgrid object
<i>num</i>	Number of grid in hierarchy

Returns

Pointer to specified grid

8.21.2.8 VEXTERNC Vgrid* Vmgrid_getGridByPoint(Vmgrid * *thee*, double *pt[3]*)

Get grid in hierarchy which contains specified point or VNULL.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vmgrid object
<i>pt</i>	Point to check

Returns

Pointer to specified grid

8.21.2.9 VEXTERNC int Vmgrid_gradient (*Vmgrid * thee*, *double pt[3]*, *double grad[3]*)

Get first derivative values at a point.

Author

Nathan Baker and Steve Bond

Parameters

<i>thee</i>	Pointer to Vmgrid object
<i>pt</i>	Location to evaluate gradient
<i>grad</i>	Gradient

Returns

1 if successful, 0 if off grid

Definition at line 176 of file [vmgrid.c](#).

8.21.2.10 VEXTERNC int Vmgrid_value (*Vmgrid * thee*, *double x[3]*, *double * value*)

Get potential value (from mesh or approximation) at a point.

Author

Nathan Baker

Parameters

<i>thee</i>	Vmgrid obejct
<i>x</i>	Point at which to evaluate potential
<i>value</i>	Value of data at point x

Returns

1 if successful, 0 if off grid

Definition at line 116 of file [vmgrid.c](#).

8.22 Vopot class

Potential oracle for Cartesian mesh data.

Data Structures

- struct **sVopot**

Electrostatic potential oracle for Cartesian mesh data.

Files

- file **vopot.h**

Potential oracle for Cartesian mesh data.

- file **vopot.c**

Class Vopot methods.

Typedefs

- typedef struct **sVopot Vopot**

Declaration of the Vopot class as the Vopot structure.

Functions

- VEXTERNC **Vopot * Vopot_ctor (Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)**

Construct Vopot object with values obtained from Vpmg_readDX (for example)

- VEXTERNC int **Vopot_ctor2 (Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)**

Initialize Vopot object with values obtained from Vpmg_readDX (for example)

- VEXTERNC int **Vopot_pot (Vopot *thee, double x[3], double *pot)**

Get potential value (from mesh or approximation) at a point.

- VEXTERNC void **Vopot_dtor (Vopot **thee)**

Object destructor.

- VEXTERNC void **Vopot_dtor2 (Vopot *thee)**

FORTRAN stub object destructor.

- VEXTERNC int **Vopot_curvature (Vopot *thee, double pt[3], int cflag, double *curv)**

Get second derivative values at a point.

- VEXTERNC int **Vopot_gradient (Vopot *thee, double pt[3], double grad[3])**

Get first derivative values at a point.

8.22.1 Detailed Description

Potential oracle for Cartesian mesh data.

8.22.2 Function Documentation

8.22.2.1 VEXTERNC Vopot* Vopot_ctor (*Vmgrid * mgrid*, *Vpbe * pbe*, *Vbcfl bcfl*)

Construct Vopot object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

<i>mgrid</i>	Multiple grid object containing potential data (in units kT/e)
<i>pbe</i>	Pointer to Vpbe object for parameters
<i>bcfl</i>	Boundary condition to use for potential values off the grid

Returns

Newly allocated and initialized Vopot object

Definition at line 67 of file [vopot.c](#).

8.22.2.2 VEXTERNC int Vopot_ctor2 (*Vopot * thee*, *Vmgrid * mgrid*, *Vpbe * pbe*, *Vbcfl bcfl*)

Initialize Vopot object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to newly allocated Vopot object
<i>mgrid</i>	Multiple grid object containing potential data (in units kT/e)
<i>pbe</i>	Pointer to Vpbe object for parameters
<i>bcfl</i>	Boundary condition to use for potential values off the grid

Returns

1 if successful, 0 otherwise

Definition at line 82 of file [vopot.c](#).

8.22.2.3 VEXTERNC int Vopot_curvature (*Vopot * thee*, double *pt[3]*, int *cflag*, double * *curv*)

Get second derivative values at a point.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vopot object
<i>pt</i>	Location to evaluate second derivative
<i>cflag</i>	<ul style="list-style-type: none"> • 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature
<i>curv</i>	Set to specified curvature value

Returns

1 if successful, 0 otherwise

Definition at line 216 of file [vopot.c](#).

8.22.2.4 VEXTERNC void Vopot_dtor (*Vopot ** thee*)

Object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location of object to be destroyed
-------------	--

Definition at line 96 of file [vopot.c](#).

8.22.2.5 VEXTERNC void Vopot_dtor2 (*Vopot * thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
-------------	-----------------------------------

Definition at line [109](#) of file [vopot.c](#).

8.22.2.6 VEXTERNC int Vopot_gradient (*Vopot *thee*, double *pt[3]*, double *grad[3]*)

Get first derivative values at a point.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vopot object
<i>pt</i>	Location to evaluate gradient
<i>grad</i>	Gradient

Returns

1 if successful, 0 otherwise

Definition at line [302](#) of file [vopot.c](#).

8.22.2.7 VEXTERNC int Vopot_pot (*Vopot *thee*, double *x[3]*, double **pot*)

Get potential value (from mesh or approximation) at a point.

Author

Nathan Baker

Parameters

<i>thee</i>	Vopot obejct
<i>x</i>	Point at which to evaluate potential
<i>pot</i>	Set to dimensionless potential (units kT/e) at point x

Returns

1 if successful, 0 otherwise

Definition at line 116 of file [vopot.c](#).

8.23 Vpmg class

A wrapper for Mike Holst's PMG multigrid code.

Data Structures

- struct [sVpmg](#)

Contains public data members for Vpmg class/module.

Files

- file [vpmg.h](#)

Contains declarations for class Vpmg.

- file [vpmg.c](#)

Class Vpmg methods.

Typedefs

- typedef struct [sVpmg](#) [Vpmg](#)

Declaration of the Vpmg class as the Vpmg structure.

Functions

- VEXTERNC unsigned long int [Vpmg_memChk](#) ([Vpmg](#) *thee)

Return the memory used by this structure (and its contents) in bytes.

- VEXTERNC [Vpmg](#) * [Vpmg_ctor](#) ([Vpmgp](#) *parms, [Vpbe](#) *pbe, int focusFlag, [Vpmg](#) *pmgOLD, [MGparm](#) *mgparm, [PBEparm_calcEnergy](#) energyFlag)

Constructor for the Vpmg class (allocates new memory)

- VEXTERNC int [Vpmg_ctor2](#) ([Vpmg](#) *thee, [Vpmgp](#) *parms, [Vpbe](#) *pbe, int focusFlag, [Vpmg](#) *pmgOLD, [MGparm](#) *mgparm, [PBEparm_calcEnergy](#) energyFlag)

FORTRAN stub constructor for the Vpmg class (uses previously-allocated memory)

- VEXTERNC void [Vpmg_dtor](#) ([Vpmg](#) **thee)

Object destructor.

- VEXTERNC void [Vpmg_dtor2](#) ([Vpmg](#) *thee)

FORTRAN stub object destructor.

- VEXTERNC int `Vpmg_fillco` (`Vpmg` *thee, `Vsurf_Meth` surfMeth, double splineWin, `Vchrg_Meth` chargeMeth, int useDielXMap, `Vgrid` *dielXMap, int useDielYMap, `Vgrid` *dielYMap, int useDielZMap, `Vgrid` *dielZMap, int useKappaMap, `Vgrid` *kappaMap, int usePotMap, `Vgrid` *potMap, int useChargeMap, `Vgrid` *chargeMap)

Fill the coefficient arrays prior to solving the equation.

- VEXTERNC int `Vpmg_solve` (`Vpmg` *thee)

Solve the PBE using PMG.

- VEXTERNC int `Vpmg_solveLaplace` (`Vpmg` *thee)

Solve Poisson's equation with a homogeneous Laplacian operator using the solvent dielectric constant. This solution is performed by a sine wave decomposition.

- VEXTERNC double `Vpmg_energy` (`Vpmg` *thee, int extFlag)

Get the total electrostatic energy.

- VEXTERNC double `Vpmg_qfEnergy` (`Vpmg` *thee, int extFlag)

Get the "fixed charge" contribution to the electrostatic energy.

- VEXTERNC double `Vpmg_qfAtomEnergy` (`Vpmg` *thee, `Vatom` *atom)

Get the per-atom "fixed charge" contribution to the electrostatic energy.

- VEXTERNC double `Vpmg_qmEnergy` (`Vpmg` *thee, int extFlag)

Get the "mobile charge" contribution to the electrostatic energy.

- VEXTERNC double `Vpmg_dielEnergy` (`Vpmg` *thee, int extFlag)

Get the "polarization" contribution to the electrostatic energy.

- VEXTERNC double `Vpmg_dielGradNorm` (`Vpmg` *thee)

Get the integral of the gradient of the dielectric function.

- VEXTERNC int `Vpmg_force` (`Vpmg` *thee, double *force, int atomID, `Vsurf_Meth` srfm, `Vchrg_Meth` chgm)

Calculate the total force on the specified atom in units of k_B T/AA.

- VEXTERNC int `Vpmg_qfForce` (`Vpmg` *thee, double *force, int atomID, `Vchrg_Meth` chgm)

Calculate the "charge-field" force on the specified atom in units of k_B T/AA.

- VEXTERNC int `Vpmg_dbForce` (`Vpmg` *thee, double *dbForce, int atomID, `Vsurf_Meth` srfm)

Calculate the dielectric boundary forces on the specified atom in units of k_B T/AA.

- VEXTERNC int `Vpmg_ibForce` (`Vpmg` *thee, double *force, int atomID, `Vsurf_Meth` srfm)

Calculate the osmotic pressure on the specified atom in units of k_B T/AA.

- VEXTERNC void `Vpmg_setPart` (`Vpmg` *thee, double lowerCorner[3], double upperCorner[3], int bflags[6])

Set partition information which restricts the calculation of observables to a (rectangular) subset of the problem domain.

- VEXTERNC void `Vpmg_unsetPart` (`Vpmg` *thee)

Remove partition restrictions.

- VEXTERNC int `Vpmg_fillArray` (`Vpmg *thee`, `double *vec`, `Vdata_Type type`, `double parm`, `Vhal_PBEType pbetype`, `PBEparm *pbeparm`)

Fill the specified array with accessibility values.
- VPUBLIC void `Vpmg_fieldSpline4` (`Vpmg *thee`, `int atomID`, `double field[3]`)

Computes the field at an atomic center using a stencil based on the first derivative of a 5th order B-spline.
- VEXTERNC double `Vpmg_qfPermanentMultipoleEnergy` (`Vpmg *thee`, `int atomID`)

Computes the permanent multipole electrostatic hydration energy (the polarization component of the hydration energy currently computed in TINKER).
- VEXTERNC void `Vpmg_qfPermanentMultipoleForce` (`Vpmg *thee`, `int atomID`, `double force[3]`, `double torque[3]`)

Computes the q-Phi Force for permanent multipoles based on 5th order B-splines.
- VEXTERNC void `Vpmg_ibPermanentMultipoleForce` (`Vpmg *thee`, `int atomID`, `double force[3]`)

Compute the ionic boundary force for permanent multipoles.
- VEXTERNC void `Vpmg_dbPermanentMultipoleForce` (`Vpmg *thee`, `int atomID`, `double force[3]`)

Compute the dielectric boundary force for permanent multipoles.
- VEXTERNC void `Vpmg_qfDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *induced`, `int atomID`, `double force[3]`, `double torque[3]`)

q-Phi direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.
- VEXTERNC void `Vpmg_qfNLDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *induced`, `*nllInduced`, `int atomID`, `double force[3]`, `double torque[3]`)

q-Phi direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B-Splines. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.
- VEXTERNC void `Vpmg_ibDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *induced`, `int atomID`, `double force[3]`)

Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.
- VEXTERNC void `Vpmg_ibNLDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *induced`, `*nllInduced`, `int atomID`, `double force[3]`)

Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.
- VEXTERNC void `Vpmg_dbDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *induced`, `int atomID`, `double force[3]`)

Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void `Vpmg_dbNLDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *nllInduced, int atomID, double force[3])`

Dielectric bounday direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void `Vpmg_qfMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nllInduced, int atomID, double force[3])`

Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void `Vpmg_ibMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nllInduced, int atomID, double force[3])`

Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void `Vpmg_dbMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nllInduced, int atomID, double force[3])`

Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void `Vpmg_printColComp (Vpmg *thee, char path[72], char title[72], char mxtype[3], int flag)`

Print out a column-compressed sparse matrix in Harwell-Boeing format.

8.23.1 Detailed Description

A wrapper for Mike Holst's PMG multigrid code.

Note

Many of the routines and macros are borrowed from the main.c driver (written by Mike Holst) provided with the PMG code.

8.23.2 Function Documentation

8.23.2.1 VEXTERNC `Vpmg* Vpmg_ctor (Vpmgp * parms, Vpbe * pbe, int focusFlag, Vpmg * pmgOLD, MGparm * mgparm, PBEparm_calcEnergy energyFlag)`

Constructor for the Vpmg class (allocates new memory)

Author

Nathan Baker

Returns

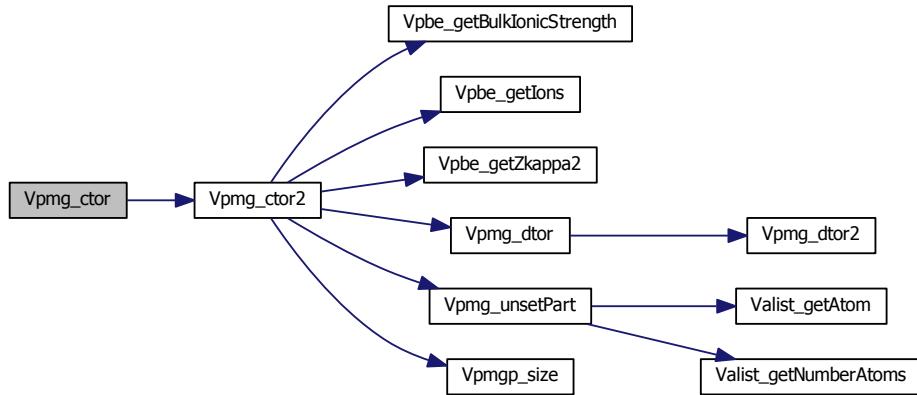
Pointer to newly allocated Vpmg object

Parameters

<i>parms</i>	PMG parameter object
<i>pbe</i>	PBE-specific variables
<i>focusFlag</i>	1 for focusing, 0 otherwise
<i>pmgOLD</i>	Old Vpmg object to use for boundary conditions
<i>mgparm</i>	MGparm parameter object for boundary conditions
<i>energyFlag</i>	What types of energies to calculate

Definition at line 127 of file [vpmg.c](#).

Here is the call graph for this function:



```

8.23.2.2 VEXTERNC int Vpmg_ctor2( Vpmg * thee, Vpmgp * parms, Vpbe * pbe, int
focusFlag, Vpmg * pmgOLD, MGparm * mgparm, PBEparm_<code>_calcEnergy</code>
energyFlag )
  
```

FORTRAN stub constructor for the Vpmg class (uses previously-allocated memory)

Author

Nathan Baker

Returns

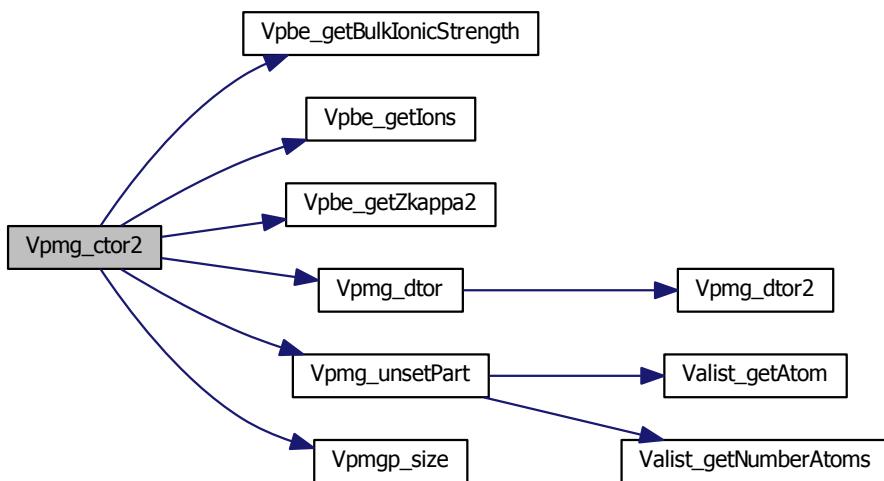
1 if successful, 0 otherwise

Parameters

<i>thee</i>	Memory location for object
<i>parms</i>	PMG parameter object
<i>pbe</i>	PBE-specific variables
<i>focusFlag</i>	1 for focusing, 0 otherwise
<i>pmgOLD</i>	Old Vpmg object to use for boundary conditions (can be VNULL if focusFlag = 0)
<i>mgparm</i>	MGparm parameter object for boundary conditions (can be VNULL if focusFlag = 0)
<i>energyFlag</i>	What types of energies to calculate (ignored if focusFlag = 0)

Definition at line 139 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.23.2.3 **VEXTERNC void Vpmg_dbDirectPolForce (*Vpmg * thee, Vgrid * perm, Vgrid * induced, int atomID, double force[3]*)**

Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>induced</i>	Induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.23.2.4 **VEXTERNC int Vpmg_dbForce (*Vpmg * thee, double * dbForce, int atomID, Vsurf_Meth srfm*)**

Calculate the dielectric boundary forces on the specified atom in units of k_B T/AA.

Author

Nathan Baker

Note

- Using the force evaluation methods of Im et al (Roux group), Comput Phys Commun, 111, 59--75 (1998). However, this gives the whole (self-interactions included) force -- reaction field forces will have to be calculated at higher level.

- No contributions are made from higher levels of focusing.

Returns

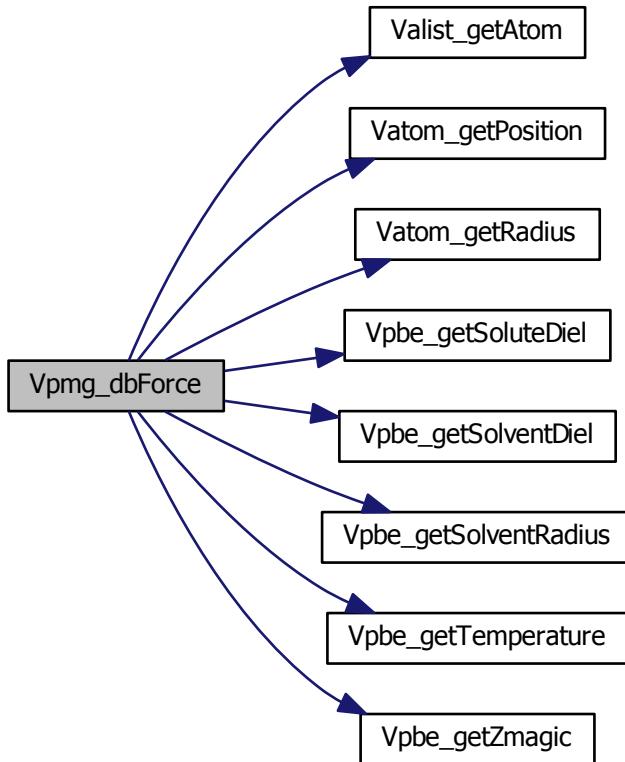
1 if successful, 0 otherwise

Parameters

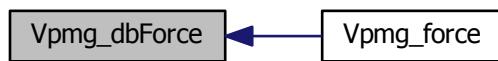
<i>thee</i>	Vpmg object
<i>dbForce</i>	3*sizeof(double) space to hold the dielectric boundary force in units of k_B T/AA
<i>atomID</i>	Valist ID of desired atom
<i>srfm</i>	Surface discretization method

Definition at line 5752 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.23.2.5 VEXTERNC void Vpmg_dbMutualPolForce (*Vpmg * thee, Vgrid * induced, Vgrid * nllInduced, int atomID, double force[3]*)

Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>induced</i>	Induced dipole potential
<i>nllInduced</i>	Non-local induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.23.2.6 VEXTERNC void Vpmg_dbNLDirectPolForce (*Vpmg * thee, Vgrid * perm, Vgrid * nllInduced, int atomID, double force[3]*)

Dielectric bounday direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>nllInduced</i>	Non-local induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.23.2.7 VEXTERNC void Vpmg_dbPermanentMultipoleForce (*Vpmg * thee, int atomID, double force[3]*)

Compute the dielectric boundary force for permanent multipoles.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.23.2.8 VEXTERNC double Vpmg_dielEnergy (Vpmg * *thee*, int *extFlag*)

Get the "polarization" contribution to the electrostatic energy.

Using the solution at the finest mesh level, get the electrostatic energy due to the interaction of the mobile charges with the potential:

$$G = \frac{1}{2} \int \epsilon (\nabla u)^2 dx$$

where epsilon is the dielectric parameter and u(x) is the dimensionless electrostatic potential. The energy is scaled to units of k_b T.

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via Vpmg_setPart and are generally useful for parallel runs.

Returns

The polarization electrostatic energy in units of k_B T.

Parameters

<i>thee</i>	Vpmg object
<i>extFlag</i>	If this was a focused calculation, include (1 -- for serial calculations) or ignore (0 -- for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1190 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.23.2.9 VEXTERNC double Vpmg_dielGradNorm (*Vpmg * thee*)

Get the integral of the gradient of the dielectric function.

Using the dielectric map at the finest mesh level, calculate the integral of the norm of the dielectric function gradient routines of Im et al (see *Vpmg_dbForce* for reference):

$$\int \|\nabla \epsilon\| dx$$

where epsilon is the dielectric parameter. The integral is returned in units of A^2.

Author

Nathan Baker restrictions on the subdomain over which it is calculated. Such limits can be set via *Vpmg_setPart* and are generally useful for parallel runs.

Returns

The integral in units of A^2.

Parameters

<i>thee</i>	Vpmg object
-------------	-------------

Definition at line 1237 of file [vpmg.c](#).

8.23.2.10 VEXTERNC void Vpmg_dtor (Vpmg ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

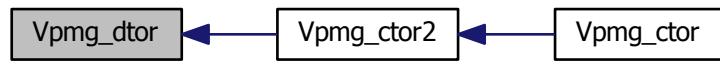
<i>thee</i>	Pointer to memory location of object to be destroyed
-------------	--

Definition at line 495 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.23.2.11 VEXTERNC void Vpmg_dtor2 (Vpmg * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
-------------	-----------------------------------

Definition at line 505 of file [vpmg.c](#).

Here is the caller graph for this function:

8.23.2.12 VEXTERNC double Vpmg_energy (Vpmg * *thee*, int *extFlag*)

Get the total electrostatic energy.

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via `Vpmg_setPart` and are generally useful for parallel runs.

Returns

The electrostatic energy in units of $k_B T$.

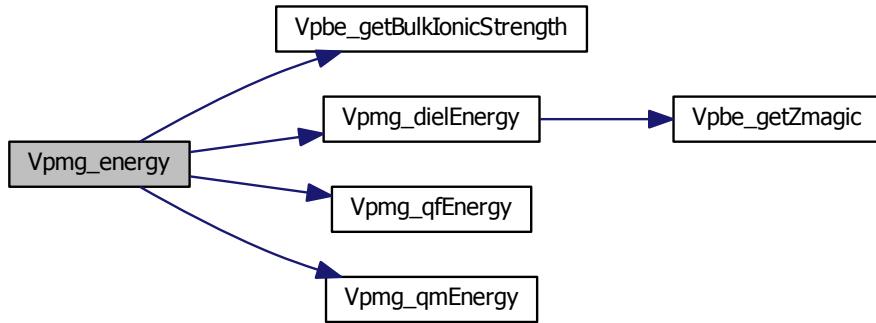
Parameters

<i>thee</i>	Vpmg object
<i>extFlag</i>	If this was a focused calculation, include (1 -- for serial calculations) or ignore (0 -- for parallel calculations) energy contributions from outside the focusing

Generated on Fri Jun 10 2017:43:49 for APBS by Doxygen

Definition at line 1161 of file [vpmg.c](#).

Here is the call graph for this function:



8.23.2.13 VPUBLIC void Vpmg_fieldSpline4 (`Vpmg * thee, int atomID, double field[3]`)

Computes the field at an atomic center using a stencil based on the first derivative of a 5th order B-spline.

Author

Michael Schnieders

Parameters

<code>thee</code>	Vpmg object
<code>atomID</code>	Atom index
<code>field</code>	The (returned) electric field

8.23.2.14 VEXTERNC int Vpmg_fillArray (`Vpmg * thee, double * vec, Vdata_Type type, double parm, Vhal_PBEType pbetype, PBEparm * pbeparm`)

Fill the specified array with accessibility values.

Author

Nathan Baker

Returns

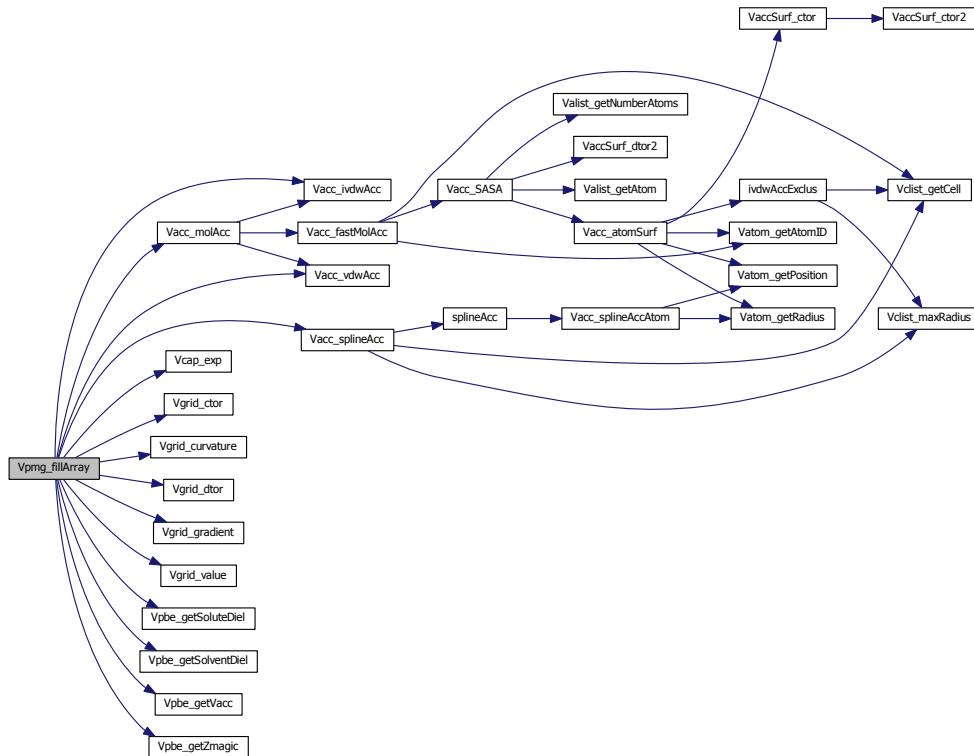
1 if successful, 0 otherwise

Parameters

<i>thee</i>	Vpmg object
<i>vec</i>	A $nx \times ny \times nz \times \text{sizeof(double)}$ array to contain the values to be written
<i>type</i>	What to write
<i>parm</i>	Parameter for data type definition (if needed)
<i>pbeptype</i>	Parameter for PBE type (if needed)
<i>pbeparm</i>	Pass in the PBE parameters (if needed)

Definition at line 826 of file vpmg.c.

Here is the call graph for this function:



8.23.2.15 VEXTERNC int Vpmg_fillco (*Vpmg * thee, Vsurf_Meth surfMeth, double splineWin, Vchrg_Meth chargeMeth, int useDielXMap, Vgrid * dielXMap, int useDielYMap, Vgrid * dielYMap, int useDielZMap, Vgrid * dielZMap, int useKappaMap, Vgrid * kappaMap, int usePotMap, Vgrid * potMap, int useChargeMap, Vgrid * chargeMap*)

Fill the coefficient arrays prior to solving the equation.

Author

Nathan Baker

Returns

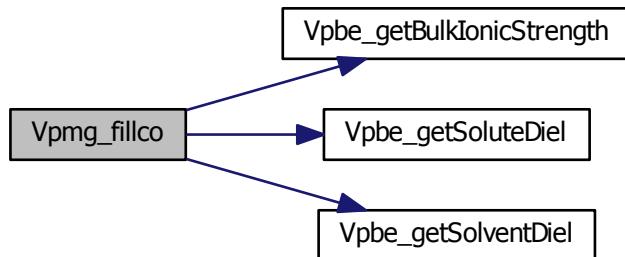
1 if successful, 0 otherwise

Parameters

<i>thee</i>	Vpmg object
<i>surfMeth</i>	Surface discretization method
<i>splineWin</i>	Spline window (in A) for surfMeth = VSM SPLINE
<i>chargeMeth</i>	Charge discretization method
<i>useDielXMap</i>	Boolean to use dielectric map argument
<i>dielXMap</i>	External dielectric map
<i>useDielYMap</i>	Boolean to use dielectric map argument
<i>dielYMap</i>	External dielectric map
<i>useDielZMap</i>	Boolean to use dielectric map argument
<i>dielZMap</i>	External dielectric map
<i>useKappa- paMap</i>	Boolean to use kappa map argument
<i>kappaMap</i>	External kappa map
<i>usePotMap</i>	Boolean to use potential map argument
<i>potMap</i>	External potential map
<i>useChargeMa- p</i>	Boolean to use charge map argument
<i>chargeMap</i>	External charge map

Definition at line 5422 of file [vpmg.c](#).

Here is the call graph for this function:



8.23.2.16 VEXTERNC int Vpmg_force (`Vpmg * thee`, `double * force`, `int atomID`, `Vsurf_Meth srfm`, `Vchrg_Meth chgm`)

Calculate the total force on the specified atom in units of k_B T/AA.

Author

Nathan Baker

Note

- Using the force evaluation methods of Im et al (Roux group), Comput Phys Commun, 111, 59--75 (1998). However, this gives the whole (self-interactions included) force -- reaction field forces will have to be calculated at higher level.
- No contributions are made from higher levels of focusing.

Returns

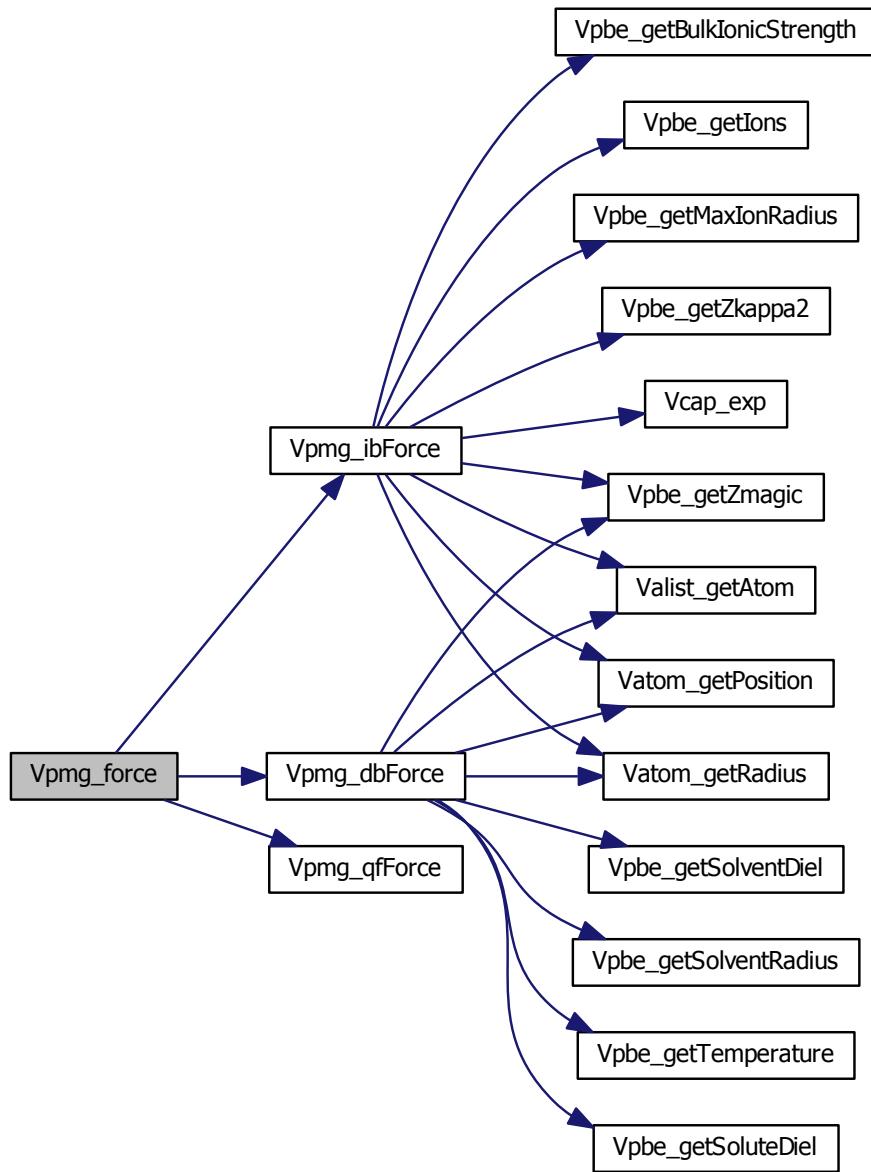
1 if successful, 0 otherwise

Parameters

<code>thee</code>	Vpmg object
<code>force</code>	3*sizeof(double) space to hold the force in units of k_B T/AA
<code>atomID</code>	Valist ID of desired atom
<code>srfm</code>	Surface discretization method
<code>chgm</code>	Charge discretization method

Definition at line [5564](#) of file [vpmg.c](#).

Here is the call graph for this function:



8.23.2.17 VEXTERNC void Vpmg_ibDirectPolForce (*Vpmg * thee, Vgrid * perm, Vgrid * induced, int atomID, double force[3]*)

Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>induced</i>	Induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.23.2.18 VEXTERNC int Vpmg_ibForce (*Vpmg * thee, double * force, int atomID, Vsurf_Meth srfm*)

Calculate the osmotic pressure on the specified atom in units of k_B T/AA.

Author

Nathan Baker

Note

- Using the force evaluation methods of Im et al (Roux group), Comput Phys Commun, 111, 59–75 (1998). However, this gives the whole (self-interactions included) force -- reaction field forces will have to be calculated at higher level.
- No contributions are made from higher levels of focusing.

Returns

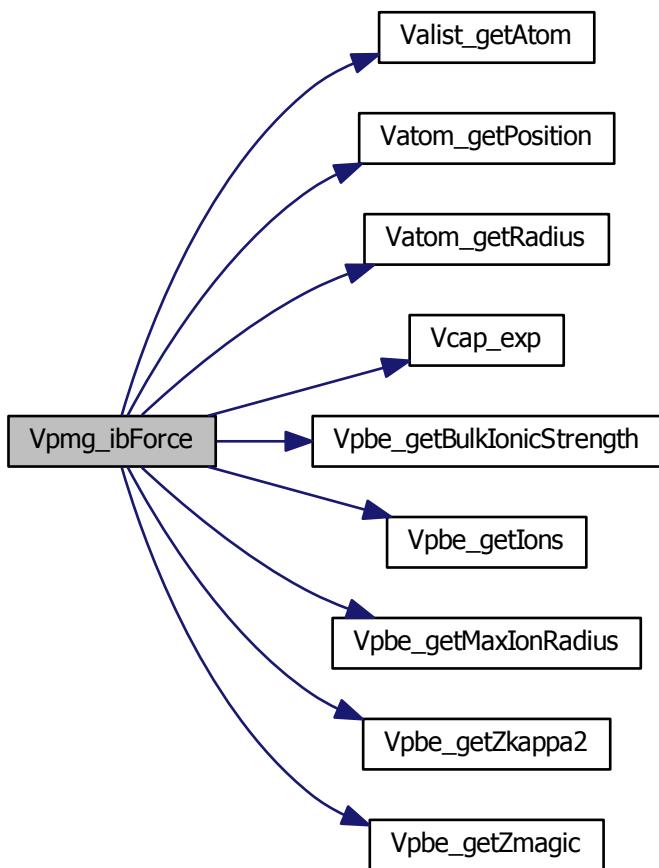
1 if successful, 0 otherwise

Parameters

<i>thee</i>	Vpmg object
<i>force</i>	3*sizeof(double) space to hold the boundary force in units of k_B T/AA
<i>atomID</i>	Valist ID of desired atom
<i>srfm</i>	Surface discretization method

Definition at line 5587 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.23.2.19 VEXTERNC void Vpmg_ibMutualPolForce (*Vpmg * thee, Vgrid * induced, Vgrid * nllInduced, int atomID, double force[3]*)

Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>induced</i>	Induced dipole potential
<i>nllInduced</i>	Non-local induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.23.2.20 VEXTERNC void Vpmg_ibNLDirectPolForce (*Vpmg * thee, Vgrid * perm, Vgrid * nllInduced, int atomID, double force[3]*)

Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>nInduced</i>	Induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.23.2.21 VEXTERNC void Vpmg_ibPermanentMultipoleForce (*Vpmg * thee*, int *atomID*, double *force[3]*)

Compute the ionic boundary force for permanent multipoles.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.23.2.22 VEXTERNC unsigned long int Vpmg_memChk (*Vpmg * thee*)

Return the memory used by this structure (and its contents) in bytes.

Author

Nathan Baker

Returns

The memory used by this structure and its contents in bytes

Parameters

<i>thee</i>	Object for memory check
-------------	-------------------------

Definition at line 66 of file [vpmg.c](#).

8.23.2.23 VEXTERNC void Vpmg_printColComp (*Vpmg * thee*, char *path[72]*, char *title[72]*, char *mxtype[3]*, int *flag*)

Print out a column-compressed sparse matrix in Harwell-Boeing format.

Author

Nathan Baker

Bug

Can this path variable be replaced with a Vio socket?

Parameters

<i>thee</i>	Vpmg object
<i>path</i>	The file to which the matrix is to be written
<i>title</i>	The title of the matrix
<i>mxtyle</i>	The type of REAL-valued matrix, a 3-character string of the form "R_A" where the '_' can be one of: <ul style="list-style-type: none">• S: symmetric matrix• U: unsymmetric matrix• H: Hermitian matrix• Z: skew-symmetric matrix• R: rectangular matrix
<i>flag</i>	The operator to compress: <ul style="list-style-type: none">• 0: Poisson operator• 1: Linearization of the full Poisson-Boltzmann operator around the current solution

Definition at line 74 of file [vpmg.c](#).

8.23.2.24 VEXTERNC double Vpmg_qfAtomEnergy (Vpmg * *thee*, Vatom * *atom*)

Get the per-atom "fixed charge" contribution to the electrostatic energy.

Using the solution at the finest mesh level, get the electrostatic energy due to the interaction of the fixed charges with the potential:

$$G = qu(r),$$

where q\$ is the charge and r is the location of the atom of interest. The result is returned in units of k_B T. Clearly, no self-interaction terms are removed. A factor a 1/2 has to be included to convert this to a real energy.

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via `Vpmg_setPart` and are generally useful for parallel runs.

Returns

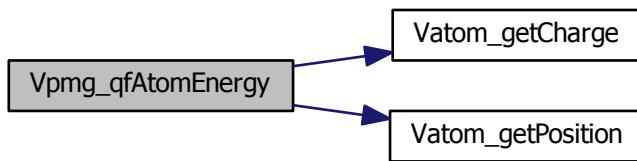
The fixed charge electrostatic energy in units of $k_B T$.

Parameters

<i>thee</i>	The Vpmg object
<i>atom</i>	The atom for energy calculations

Definition at line 1620 of file [vpmg.c](#).

Here is the call graph for this function:



8.23.2.25 VEXTERNC void Vpmg_qfDirectPolForce (Vpmg * *thee*, Vgrid * *perm*, Vgrid * *induced*, int *atomID*, double *force*[3], double *torque*[3])

q-Phi direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential

<i>induced</i>	Induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force
<i>torque</i>	(returned) torque

8.23.2.26 VEXTERNC double Vpmg_qfEnergy (*Vpmg * thee, int extFlag*)

Get the "fixed charge" contribution to the electrostatic energy.

Using the solution at the finest mesh level, get the electrostatic energy due to the interaction of the fixed charges with the potential:

$$G = \sum_i q_i u(r_i)$$

and return the result in units of k_B T. Clearly, no self-interaction terms are removed. A factor a 1/2 has to be included to convert this to a real energy.

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via *Vpmg_setPart* and are generally useful for parallel runs.

Returns

The fixed charge electrostatic energy in units of k_B T.

Parameters

<i>thee</i>	<i>Vpmg</i> object
<i>extFlag</i>	If this was a focused calculation, include (1 -- for serial calculations) or ignore (0 -- for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1520 of file [vpmg.c](#).

Here is the caller graph for this function:



8.23.2.27 VEXTERNC int Vpmg_qfForce (*Vpmg * thee, double * force, int atomID, Vchrg_Meth chgm*)

Calculate the "charge-field" force on the specified atom in units of k_B T/AA.

Author

Nathan Baker

Note

- Using the force evaluation methods of Im et al (Roux group), Comput Phys Commun, 111, 59--75 (1998). However, this gives the whole (self-interactions included) force -- reaction field forces will have to be calculated at higher level.
- No contributions are made from higher levels of focusing.

Returns

1 if sucessful, 0 otherwise

Parameters

<i>thee</i>	Vpmg object
<i>force</i>	3*sizeof(double) space to hold the force in units of k_B T/A
<i>atomID</i>	Valist ID of desired atom
<i>chgm</i>	Charge discretization method

Definition at line 6009 of file [vpmg.c](#).

Here is the caller graph for this function:



8.23.2.28 VEXTERNC void Vpmg_qfMutualPolForce (*Vpmg * thee, Vgrid * induced, Vgrid * nllInduced, int atomID, double force[3]*)

Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>induced</i>	Induced dipole potential
<i>nllInduced</i>	Non-local induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.23.2.29 VEXTERNC void Vpmg_qfNLDirectPolForce (*Vpmg * thee, Vgrid * perm, Vgrid * nllInduced, int atomID, double force[3], double torque[3]*)

q-Phi direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B-Splines. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>nllInduced</i>	Non-local induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force
<i>torque</i>	(returned) torque

8.23.2.30 VEXTERNC double Vpmg_qfPermanentMultipoleEnergy (*Vpmg * thee, int atomID*)

Computes the permanent multipole electrostatic hydration energy (the polarization component of the hydration energy currently computed in TINKER).

Author

Michael Schnieders

Returns

The permanent multipole electrostatic hydration energy

Parameters

<i>thee</i>	Vpmg object
<i>atomID</i>	Atom index

8.23.2.31 VEXTERNC void Vpmg_qfPermanentMultipoleForce (*Vpmg * thee, int atomID, double force[3], double torque[3]*)

Computes the q-Phi Force for permanent multipoles based on 5th order B-splines.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>atomID</i>	Atom index
<i>force</i>	(returned) force
<i>torque</i>	(returned) torque

8.23.2.32 VEXTERNC double Vpmg_qmEnergy (*Vpmg * thee, int extFlag*)

Get the "mobile charge" contribution to the electrostatic energy.

Using the solution at the finest mesh level, get the electrostatic energy due to the interaction of the mobile charges with the potential:

$$G = \frac{1}{4I_s} \sum_i c_i q_i^2 \int \kappa^2(x) e^{-q_i u(x)} dx$$

for the NPBE and

$$G = \frac{1}{2} \int \bar{\kappa}^2(x) u^2(x) dx$$

for the LPBE. Here i denotes the counterion species, I_s is the bulk ionic strength, $\kappa^2(x)$ is the modified Debye-Huckel parameter, c_i is the concentration of species i, q_i is the charge of species i, and $u(x)$ is the dimensionless electrostatic potential. The energy is scaled to units of $k_B T$.

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via `Vpmg_setPart` and are generally useful for parallel runs.

Returns

The mobile charge electrostatic energy in units of $k_B T$.

Parameters

<i>thee</i>	Vpmg object
<i>extFlag</i>	If this was a focused calculation, include (1 -- for serial calculations) or ignore (0 -- for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1281 of file [vpmg.c](#).

Here is the caller graph for this function:



8.23.2.33 VEXTERNC void Vpmg_setPart (*Vpmg * thee*, double *lowerCorner[3]*, double *upperCorner[3]*, int *bflags[6]*)

Set partition information which restricts the calculation of observables to a (rectangular) subset of the problem domain.

Author

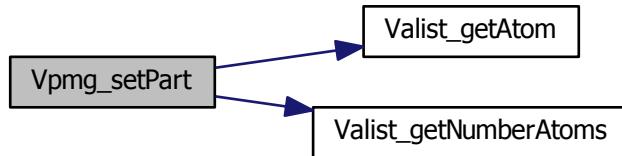
Nathan Baker

Parameters

<i>thee</i>	Vpmg object
<i>lowerCorner</i>	Partition lower corner
<i>upperCorner</i>	Partition upper corner
<i>bflags</i>	Booleans indicating whether a particular processor is on the boundary with another partition. 0 if the face is not bounded (next to) another partition, and 1 otherwise.

Definition at line 561 of file [vpmg.c](#).

Here is the call graph for this function:



8.23.2.34 VEXTERNC int Vpmg_solve (*Vpmg* * *thee*)

Solve the PBE using PMG.

Author

Nathan Baker

Returns

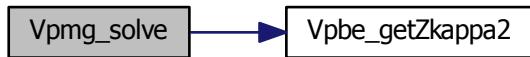
1 if successful, 0 otherwise

Parameters

<i>thee</i>	Vpmg object
-------------	-------------

Definition at line 359 of file [vpmg.c](#).

Here is the call graph for this function:



8.23.2.35 VEXTERNC int Vpmg_solveLaplace (Vpmg * *thee*)

Solve Poisson's equation with a homogeneous Laplacian operator using the solvent dielectric constant. This solution is performed by a sine wave decomposition.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Note

This function is really only for testing purposes as the PMG multigrid solver can solve the homogeneous system much more quickly. Perhaps we should implement an FFT version at some point...

Parameters

<i>thee</i>	Vpmg object
-------------	-------------

Definition at line [6784](#) of file [vpmg.c](#).

Here is the call graph for this function:

**8.23.2.36 VEXTERNC void Vpmg_unsetPart (Vpmg * *thee*)**

Remove partition restrictions.

Author

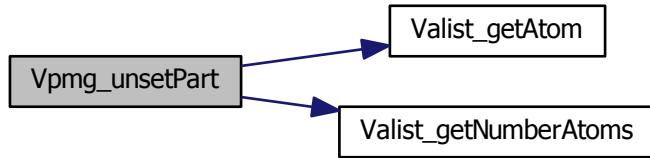
Nathan Baker

Parameters

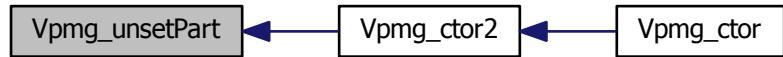
	<i>thee</i> Vpmg object
--	---------------------------

Definition at line 806 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.24 Vpmgp class

Parameter structure for Mike Holst's PMGP code.

Data Structures

- struct [sVpmgp](#)

Contains public data members for Vpmgp class/module.

Files

- file [vpmgp.h](#)
Contains declarations for class Vpmgp.
- file [vpmgp.c](#)
Class Vpmgp methods.

Typedefs

- typedef struct [sVpmgp](#) [Vpmgp](#)
Declaration of the Vpmgp class as the sVpmgp structure.

Functions

- VEXTERNC [Vpmgp * Vpmgp_ctor](#) ([MGparm](#) *mgparm)
Construct PMG parameter object and initialize to default values.
- VEXTERNC int [Vpmgp_ctor2](#) ([Vpmgp](#) *thee, [MGparm](#) *mgparm)
FORTRAN stub to construct PMG parameter object and initialize to default values.
- VEXTERNC void [Vpmgp_dtor](#) ([Vpmgp](#) **thee)
Object destructor.
- VEXTERNC void [Vpmgp_dtor2](#) ([Vpmgp](#) *thee)
FORTRAN stub for object destructor.
- VEXTERNC void [Vpmgp_size](#) ([Vpmgp](#) *thee)
Determine array sizes and parameters for multigrid solver.
- VEXTERNC void [Vpmgp_makeCoarse](#) (int numLevel, int nxOld, int nyOld, int nzOld, int *nxNew, int *nyNew, int *nzNew)
Coarsen the grid by the desired number of levels and determine the resulting numbers of grid points.

8.24.1 Detailed Description

Parameter structure for Mike Holst's PMGP code.

Note

Variables and many default values taken directly from PMG

8.24.2 Function Documentation

8.24.2.1 VEXTERNC `Vpmgp* Vpmgp_ctor (MGparm * mgparm)`

Construct PMG parameter object and initialize to default values.

Author

Nathan Baker

Parameters

<code>mgparm</code>	MGParm object containing parameters to be used in setup
---------------------	---

Returns

Newly allocated and initialized Vpmgp object

Definition at line 78 of file [vpmgp.c](#).

8.24.2.2 VEXTERNC int `Vpmgp_ctor2 (Vpmgp * thee, MGparm * mgparm)`

FORTRAN stub to construct PMG parameter object and initialize to default values.

Author

Nathan Baker

Parameters

<code>thee</code>	Newly allocated PMG object
<code>mgparm</code>	MGParm object containing parameters to be used in setup

Returns

1 if successful, 0 otherwise

Definition at line 95 of file [vpmgp.c](#).

8.24.2.3 VEXTERNC void `Vpmgp_dtor (Vpmgp ** thee)`

Object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory location for Vpmgp object
-------------	---

Definition at line 180 of file [vpmgp.c](#).

8.24.2.4 VEXTERNC void Vpmgp_dtor2 (Vpmgp * *thee*)

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vpmgp object
-------------	-------------------------

Definition at line 195 of file [vpmgp.c](#).

8.24.2.5 VEXTERNC void Vpmgp_makeCoarse (int *numLevel*, int *nxOld*, int *nyOld*, int *nzOld*, int * *nxNew*, int * *nyNew*, int * *nzNew*)

Coarsen the grid by the desired number of levels and determine the resulting numbers of grid points.

Author

Mike Holst and Nathan Baker

Parameters

<i>numLevel</i>	Number of levels to coarsen
<i>nxOld</i>	Number of old grid points in this direction
<i>nyOld</i>	Number of old grid points in this direction
<i>nzOld</i>	Number of old grid points in this direction
<i>nxNew</i>	Number of new grid points in this direction
<i>nyNew</i>	Number of new grid points in this direction
<i>nzNew</i>	Number of new grid points in this direction

Definition at line 314 of file [vpmgp.c](#).

8.24.2.6 VEXTERNC void Vpmgp_size (Vpmgp * *thee*)

Determine array sizes and parameters for multigrid solver.

Author

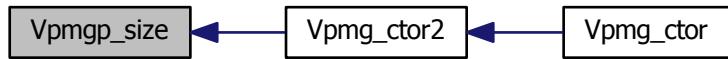
Mike Holst and Nathan Baker

Parameters

<i>thee</i>	Object to be sized
-------------	--------------------

Definition at line 198 of file [vpmgp.c](#).

Here is the caller graph for this function:



Chapter 9

Data Structure Documentation

9.1 sAPOLparm Struct Reference

Parameter structure for APOL-specific variables from input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/apolparm.h>
```

Data Fields

- int `parsed`
- double `grid` [3]
- int `setgrid`
- int `molid`
- int `setmolid`
- double `bconc`
- int `setbconc`
- double `sdens`
- int `setsdens`
- double `dpos`
- int `setdpos`
- double `press`
- int `setpress`
- `Vsurf_Meth` `srfm`
- int `setsrfm`
- double `srad`
- int `setsrad`
- double `swin`

- int `setswin`
- double `temp`
- int `settemp`
- double `gamma`
- int `setgamma`
- `APOLparm_calcEnergy calcenergy`
- int `setcalcenergy`
- `APOLparm_calcForce calcforce`
- int `setcalcforce`
- double `watsigma`
- double `watepsilon`
- double `sasa`
- double `sav`
- double `wcaEnergy`
- double `totForce` [3]
- int `setwat`

9.1.1 Detailed Description

Parameter structure for APOL-specific variables from input files.

Author

David Gohara

Definition at line 126 of file `apolparm.h`.

9.1.2 Field Documentation

9.1.2.1 double `bconc`

Vacc sphere density

Definition at line 136 of file `apolparm.h`.

9.1.2.2 `APOLparm_calcEnergy calcenergy`

Energy calculation flag

Definition at line 164 of file `apolparm.h`.

9.1.2.3 APOLparm_calcForce calcforce

Atomic forces calculation

Definition at line [167](#) of file [apolparm.h](#).

9.1.2.4 double dpos

Atom position offset

Definition at line [142](#) of file [apolparm.h](#).

9.1.2.5 double gamma

Surface tension for apolar energies/forces (in kJ/mol/A²)

Definition at line [160](#) of file [apolparm.h](#).

9.1.2.6 double grid[3]

Grid spacing

Definition at line [130](#) of file [apolparm.h](#).

9.1.2.7 int molid

Molecule ID to perform calculation on

Definition at line [133](#) of file [apolparm.h](#).

9.1.2.8 int parsed

Flag: Has this structure been filled with anything other than the default values? (0 = no, 1 = yes)

Definition at line [128](#) of file [apolparm.h](#).

9.1.2.9 double press

Solvent pressure

Definition at line [145](#) of file [apolparm.h](#).

9.1.2.10 double sasa

Solvent accessible surface area for this calculation

Definition at line 172 of file [apolparm.h](#).

9.1.2.11 double sav

Solvent accessible volume for this calculation

Definition at line 173 of file [apolparm.h](#).

9.1.2.12 double sdens

Vacc sphere density

Definition at line 139 of file [apolparm.h](#).

9.1.2.13 int setbconc

Flag,

See also

[bconc](#)

Definition at line 137 of file [apolparm.h](#).

9.1.2.14 int setcalcenergy

Flag,

See also

[calcenergy](#)

Definition at line 165 of file [apolparm.h](#).

9.1.2.15 int setcalcforce

Flag,

See also

[calcforce](#)

Definition at line 168 of file [apolparm.h](#).

9.1.2.16 int setdpos

Flag,

See also

[dpos](#)

Definition at line [143](#) of file [apolparm.h](#).

9.1.2.17 int setgamma

Flag,

See also

[gamma](#)

Definition at line [162](#) of file [apolparm.h](#).

9.1.2.18 int setgrid

Flag,

See also

[grid](#)

Definition at line [131](#) of file [apolparm.h](#).

9.1.2.19 int setmolid

Flag,

See also

[molid](#)

Definition at line [134](#) of file [apolparm.h](#).

9.1.2.20 int setpress

Flag,

See also[press](#)

Definition at line 146 of file [apolparm.h](#).

9.1.2.21 int setsdens

Flag,

See also[sdens](#)

Definition at line 140 of file [apolparm.h](#).

9.1.2.22 int setsrad

Flag,

See also[srad](#)

Definition at line 152 of file [apolparm.h](#).

9.1.2.23 int setsrfm

Flag,

See also[srfm](#)

Definition at line 149 of file [apolparm.h](#).

9.1.2.24 int setswin

Flag,

See also[swin](#)

Definition at line 155 of file [apolparm.h](#).

9.1.2.25 int settemp

Flag,

See also

[temp](#)

Definition at line [158](#) of file [apolparm.h](#).

9.1.2.26 int setwat

Boolean for determining if a water parameter is supplied. Yes = 1, No = 0

Definition at line [177](#) of file [apolparm.h](#).

9.1.2.27 double srad

Solvent radius

Definition at line [151](#) of file [apolparm.h](#).

9.1.2.28 Vsurf_Meth srfm

Surface calculation method

Definition at line [148](#) of file [apolparm.h](#).

9.1.2.29 double swin

Cubic spline window

Definition at line [154](#) of file [apolparm.h](#).

9.1.2.30 double temp

Temperature (in K)

Definition at line [157](#) of file [apolparm.h](#).

9.1.2.31 double totForce[3]

Total forces on x, y, z

Definition at line [175](#) of file [apolparm.h](#).

9.1.2.32 double watepsilon

Water oxygen Lennard-Jones well depth (kJ/mol)

Definition at line 171 of file [apolparm.h](#).

9.1.2.33 double watsigma

Water oxygen Lennard-Jones radius (Å)

Definition at line 170 of file [apolparm.h](#).

9.1.2.34 double wcaEnergy

wcaEnergy

Definition at line 174 of file [apolparm.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/apolparm.h

9.2 sFEMparm Struct Reference

Parameter structure for FEM-specific variables from input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/femparm.h>
```

Data Fields

- int [parsed](#)
- [FEMparm_CalcType](#) type
- int [settype](#)
- double [glen](#) [3]
- int [setglen](#)
- double [etol](#)
- int [setetol](#)
- [FEMparm_EtolType](#) [ekey](#)
- int [setekey](#)
- [FEMparm_EstType](#) [akeyPRE](#)
- int [setakeyPRE](#)

- `FEMparm_EstType akeySOLVE`
- `int setakeySOLVE`
- `int targetNum`
- `int settargetNum`
- `double targetRes`
- `int settargetRes`
- `int maxsolve`
- `int setmaxsolve`
- `int maxvert`
- `int setmaxvert`
- `int pkey`
- `int useMesh`
- `int meshID`

9.2.1 Detailed Description

Parameter structure for FEM-specific variables from input files.

Author

Nathan Baker

Definition at line [130](#) of file `femparm.h`.

9.2.2 Field Documentation

9.2.2.1 FEMparm_EstType akeyPRE

Adaptive refinement error estimator method for pre-solution refine. Note, this should either be FRT_UNIF or FRT_GEOM.

Definition at line [145](#) of file `femparm.h`.

9.2.2.2 FEMparm_EstType akeySOLVE

Adaptive refinement error estimator method for a posteriori solution-based refinement.

Definition at line [149](#) of file `femparm.h`.

9.2.2.3 FEMparm_EtolType ekey

Adaptive refinement interpretation of error tolerance

Definition at line [142](#) of file `femparm.h`.

9.2.2.4 double etol

Error tolerance for refinement; interpretation depends on the adaptive refinement method chosen

Definition at line 139 of file [femparm.h](#).

9.2.2.5 double glen[3]

Domain side lengths (in Å)

Definition at line 137 of file [femparm.h](#).

9.2.2.6 int maxsolve

Maximum number of solve-estimate-refine cycles

Definition at line 162 of file [femparm.h](#).

9.2.2.7 int maxvert

Maximum number of vertices in mesh (ignored if less than zero)

Definition at line 164 of file [femparm.h](#).

9.2.2.8 int meshID

External finite element mesh ID (if used)

Definition at line 171 of file [femparm.h](#).

9.2.2.9 int parsed

Flag: Has this structure been filled with anything other than * the default values? (0 = no, 1 = yes)

Definition at line 132 of file [femparm.h](#).

9.2.2.10 int pkey

Boolean sets the pkey type for going into AM_Refine pkey = 0 for non-HB based methods pkey = 1 for HB based methods

Definition at line 167 of file [femparm.h](#).

9.2.2.11 int setakeyPRE

Boolean

Definition at line [148](#) of file [femparm.h](#).

9.2.2.12 int setakeySOLVE

Boolean

Definition at line [151](#) of file [femparm.h](#).

9.2.2.13 int setekey

Boolean

Definition at line [144](#) of file [femparm.h](#).

9.2.2.14 int setetol

Boolean

Definition at line [141](#) of file [femparm.h](#).

9.2.2.15 int setglen

Boolean

Definition at line [138](#) of file [femparm.h](#).

9.2.2.16 int setmaxsolve

Boolean

Definition at line [163](#) of file [femparm.h](#).

9.2.2.17 int setmaxvert

Boolean

Definition at line [166](#) of file [femparm.h](#).

9.2.2.18 int settargetNum

Boolean

Definition at line 156 of file [femparm.h](#).

9.2.2.19 int settargetRes

Boolean

Definition at line 161 of file [femparm.h](#).

9.2.2.20 int settype

Boolean

Definition at line 136 of file [femparm.h](#).

9.2.2.21 int targetNum

Initial mesh will continue to be marked and refined with the method specified by akeyPRE until the mesh contains this many vertices or until targetRes is reached.

Definition at line 152 of file [femparm.h](#).

9.2.2.22 double targetRes

Initial mesh will continue to be marked and refined with the method specified by akeyPRE until the mesh contains no markable simplices with longest edges above this size or until targetNum is reached.

Definition at line 157 of file [femparm.h](#).

9.2.2.23 FEMparm_CalcType type

Calculation type

Definition at line 135 of file [femparm.h](#).

9.2.2.24 int useMesh

Indicates whether we use external finite element mesh

Definition at line 170 of file [femparm.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[femparm.h](#)

9.3 sMGparm Struct Reference

Parameter structure for MG-specific variables from input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/mgparm.h>
```

Data Fields

- MGparm_CalcType type
- int parsed
- int dime [3]
- int setdime
- Vchrg_Meth chgm
- int setchgm
- Vchrg_Src chgs
- int nlev
- int setnlev
- double etol
- int setetol
- double grid [3]
- int setgrid
- double glen [3]
- int setglen
- MGparm_CentMeth cmeth
- double center [3]
- int centmol
- int setcent
- double cglens [3]
- int setcglens
- double fglen [3]
- int setfglen
- MGparm_CentMeth ccmeth
- double ccenter [3]
- int ccentmol
- int setccent
- MGparm_CentMeth fcmeth

- double `fcenter` [3]
- int `fcentmol`
- int `setfgcent`
- double `partDisjCenter` [3]
- double `partDisjLength` [3]
- int `partDisjOwnSide` [6]
- int `pdime` [3]
- int `setpdime`
- int `proc_rank`
- int `setrank`
- int `proc_size`
- int `setszie`
- double `ofrac`
- int `setofrac`
- int `async`
- int `setasync`
- int `nonlintype`
- int `setnonlintype`
- int `method`
- int `setmethod`
- int `useAqua`
- int `setUseAqua`

9.3.1 Detailed Description

Parameter structure for MG-specific variables from input files.

Author

Nathan Baker and Todd Dolinsky

Note

If you add/delete/change something in this class, the member functions -- especially `MGparm_copy` -- must be modified accordingly

Definition at line 111 of file [mgparm.h](#).

9.3.2 Field Documentation

9.3.2.1 int `async`

Processor ID for asynchronous calculation

Definition at line 183 of file [mgparm.h](#).

9.3.2.2 double ccenter[3]

Coarse grid center.

Definition at line 154 of file [mgparm.h](#).

9.3.2.3 int ccentmol

Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.

Definition at line 155 of file [mgparm.h](#).

9.3.2.4 MGparm_CentMeth ccmeth

Coarse grid centering method

Definition at line 153 of file [mgparm.h](#).

9.3.2.5 double center[3]

Grid center. If ispart = 0, then this is only meaningful if cmeth = 0. However, if ispart = 1 and cmeth = MCM_PNT, then this is the center of the non-disjoint (overlapping) partition. If ispart = 1 and cmeth = MCM_MOL, then this is the vector that must be added to the center of the molecule to give the center of the non-disjoint partition.

Definition at line 135 of file [mgparm.h](#).

9.3.2.6 int centmol

Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.

Definition at line 143 of file [mgparm.h](#).

9.3.2.7 double cglen[3]

Coarse grid side lengths

Definition at line 149 of file [mgparm.h](#).

9.3.2.8 Vchrg_Meth chgm

Charge discretization method

Definition at line 119 of file [mgparm.h](#).

9.3.2.9 Vchrg_Src chgs

Charge source (Charge, Multipole, Induced Dipole, NL Induced

Definition at line 121 of file [mgparm.h](#).

9.3.2.10 MGparm_CentMeth cmeth

Centering method

Definition at line 134 of file [mgparm.h](#).

9.3.2.11 int dime[3]

Grid dimensions

Definition at line 117 of file [mgparm.h](#).

9.3.2.12 double etol

User-defined error tolerance

Definition at line 128 of file [mgparm.h](#).

9.3.2.13 double fcenter[3]

Fine grid center.

Definition at line 160 of file [mgparm.h](#).

9.3.2.14 int fcenmol

Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.

Definition at line 161 of file [mgparm.h](#).

9.3.2.15 MGparm_CentMeth fcmeth

Fine grid centering method

Definition at line 159 of file [mgparm.h](#).

9.3.2.16 double fglen[3]

Fine grid side lengths

Definition at line 151 of file [mgparm.h](#).

9.3.2.17 double glen[3]

Grid side lengths.

Definition at line 132 of file [mgparm.h](#).

9.3.2.18 double grid[3]

Grid spacings

Definition at line 130 of file [mgparm.h](#).

9.3.2.19 int method

Solver Method

Definition at line 189 of file [mgparm.h](#).

9.3.2.20 int nlev

Levels in multigrid hierarchy

Deprecated

Just ignored now

Definition at line 125 of file [mgparm.h](#).

9.3.2.21 int nonlintype

Linearity Type Method to be used

Definition at line 186 of file [mgparm.h](#).

9.3.2.22 double ofrac

Overlap fraction between procs

Definition at line 181 of file [mgparm.h](#).

9.3.2.23 int parsed

Has this structure been filled? (0 = no, 1 = yes)

Definition at line 114 of file [mgparm.h](#).

9.3.2.24 double partDisjCenter[3]

This gives the center of the disjoint partitions

Definition at line 168 of file [mgparm.h](#).

9.3.2.25 double partDisjLength[3]

This gives the lengths of the disjoint partitions

Definition at line 170 of file [mgparm.h](#).

9.3.2.26 int partDisjOwnSide[6]

Tells whether the boundary points are ours (1) or not (0)

Definition at line 172 of file [mgparm.h](#).

9.3.2.27 int pdime[3]

Grid of processors to be used in calculation

Definition at line 175 of file [mgparm.h](#).

9.3.2.28 int proc_rank

Rank of this processor

Definition at line 177 of file [mgparm.h](#).

9.3.2.29 int proc_size

Total number of processors

Definition at line 179 of file [mgparm.h](#).

9.3.2.30 int setasync

Flag,

See also

[asynch](#)

Definition at line [184](#) of file [mgparm.h](#).

9.3.2.31 int setcgcent

Flag,

See also

[ccmeth](#)

Definition at line [158](#) of file [mgparm.h](#).

9.3.2.32 int setcglen

Flag,

See also

[cglen](#)

Definition at line [150](#) of file [mgparm.h](#).

9.3.2.33 int setchgm

Flag,

See also

[chgm](#)

Definition at line [120](#) of file [mgparm.h](#).

9.3.2.34 int setdime

Flag,

See also[dime](#)

Definition at line 118 of file [mgparm.h](#).

9.3.2.35 int setetol

Flag,

See also[etol](#)

Definition at line 129 of file [mgparm.h](#).

9.3.2.36 int setfgcent

Flag,

See also[fcmeth](#)

Definition at line 164 of file [mgparm.h](#).

9.3.2.37 int setfglen

Flag,

See also[fglen](#)

Definition at line 152 of file [mgparm.h](#).

9.3.2.38 int setgcent

Flag,

See also[cmeth](#)

Definition at line 146 of file [mgparm.h](#).

9.3.2.39 int setglen

Flag,

See also

[glen](#)

Definition at line [133](#) of file [mgparm.h](#).

9.3.2.40 int setgrid

Flag,

See also

[grid](#)

Definition at line [131](#) of file [mgparm.h](#).

9.3.2.41 int setmethod

Flag,

See also

[method](#)

Definition at line [190](#) of file [mgparm.h](#).

9.3.2.42 int setnlev

Flag,

See also

[nlev](#)

Definition at line [127](#) of file [mgparm.h](#).

9.3.2.43 int setnonlintype

Flag,

See also[nonlintype](#)

Definition at line 187 of file [mgparm.h](#).

9.3.2.44 int setofrac

Flag,

See also[ofrac](#)

Definition at line 182 of file [mgparm.h](#).

9.3.2.45 int setpdime

Flag,

See also[pdime](#)

Definition at line 176 of file [mgparm.h](#).

9.3.2.46 int setrank

Flag,

See also[proc_rank](#)

Definition at line 178 of file [mgparm.h](#).

9.3.2.47 int setsize

Flag,

See also[proc_size](#)

Definition at line 180 of file [mgparm.h](#).

9.3.2.48 int setUseAqua

Flag,

See also[useAqua](#)

Definition at line 193 of file [mgparm.h](#).

9.3.2.49 MGparm_CalcType type

What type of MG calculation?

Definition at line 113 of file [mgparm.h](#).

9.3.2.50 int useAqua

Enable use of lpbe/aqua

Definition at line 192 of file [mgparm.h](#).

The documentation for this struct was generated from the following file:

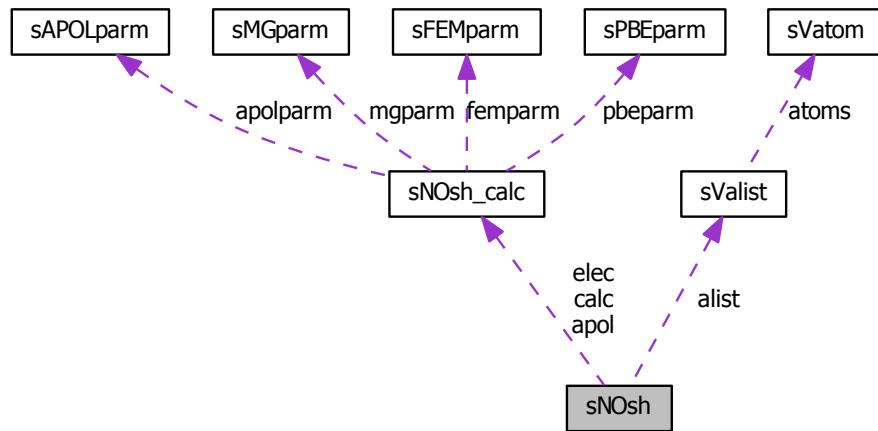
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[mgparm.h](#)

9.4 sNOsh Struct Reference

Class for parsing fixed format input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/nosh.h>
```

Collaboration diagram for sNOsh:



Data Fields

- `NOsh_calc * calc` [NOSH_MAXCALC]
- int `ncalc`
- `NOsh_calc * elec` [NOSH_MAXCALC]
- int `nelec`
- `NOsh_calc * apol` [NOSH_MAXCALC]
- int `napol`
- int `ispara`
- int `proc_rank`
- int `proc_size`
- int `bogus`
- int `elec2calc` [NOSH_MAXCALC]
- int `apol2calc` [NOSH_MAXCALC]
- int `nmol`
- char `molpath` [NOSH_MAXMOL][VMAX_ARGLEN]
- `NOsh_MolFormat molfmt` [NOSH_MAXMOL]
- `Valist * alist` [NOSH_MAXMOL]
- int `gotparm`
- char `parmpath` [VMAX_ARGLEN]

- NOsh_ParmFormat parmfmt
- int ndiel
- char dielXpath [NOSH_MAXMOL][VMAX_ARGLEN]
- char dielYpath [NOSH_MAXMOL][VMAX_ARGLEN]
- char dielZpath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format dielfmt [NOSH_MAXMOL]
- int nkappa
- char kappapath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format kappafmt [NOSH_MAXMOL]
- int npot
- char potpath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format potfmt [NOSH_MAXMOL]
- int ncharge
- char chargepath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format chargefmt [NOSH_MAXMOL]
- int nmesh
- char meshpath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format meshfmt [NOSH_MAXMOL]
- int nprint
- NOsh_PrintType printwhat [NOSH_MAXPRINT]
- int printnarg [NOSH_MAXPRINT]
- int printcalc [NOSH_MAXPRINT][NOSH_MAXPOP]
- int printop [NOSH_MAXPRINT][NOSH_MAXPOP]
- int parsed
- char elecname [NOSH_MAXCALC][VMAX_ARGLEN]
- char apolname [NOSH_MAXCALC][VMAX_ARGLEN]

9.4.1 Detailed Description

Class for parsing fixed format input files.

Author

Nathan Baker

Definition at line 182 of file [nosh.h](#).

9.4.2 Field Documentation

9.4.2.1 Valist* alist[NOSH_MAXMOL]

Molecules for calculation (can be used in setting mesh centers

Definition at line 221 of file [nosh.h](#).

9.4.2.2 NOsh_calc* apol[NOSH_MAXCALC]

The array of calculation objects corresponding to APOLAR statements read in the input file. Compare to [sNOsh::calc](#)

Definition at line [195](#) of file [nosh.h](#).

9.4.2.3 int apol2calc[NOSH_MAXCALC]

(see elec2calc)

Definition at line [216](#) of file [nosh.h](#).

9.4.2.4 char apolname[NOSH_MAXCALC][VMAX_ARGLEN]

Optional user-specified name for APOLAR statement

Definition at line [256](#) of file [nosh.h](#).

9.4.2.5 int bogus

A flag which tells routines using NOsh that this particular NOsh is broken -- useful for parallel focusing calculations where the user gave us too many processors (1 => ignore this NOsh; 0 => this NOsh is OK)

Definition at line [204](#) of file [nosh.h](#).

9.4.2.6 NOsh_calc* calc[NOSH_MAXCALC]

The array of calculation objects corresponding to actual calculations performed by the code. Compare to [sNOsh::elec](#)

Definition at line [184](#) of file [nosh.h](#).

9.4.2.7 Vdata_Format chargefmt[NOSH_MAXMOL]

Charge maps fileformats

Definition at line [242](#) of file [nosh.h](#).

9.4.2.8 char chargepath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to charge map files

Definition at line [241](#) of file [nosh.h](#).

9.4.2.9 Vdata_Format dielfmt[NOSH_MAXMOL]

Dielectric maps file formats

Definition at line [233](#) of file [nosh.h](#).

9.4.2.10 char dielXpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to x-shifted dielectric map files

Definition at line [227](#) of file [nosh.h](#).

9.4.2.11 char dielYpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to y-shifted dielectric map files

Definition at line [229](#) of file [nosh.h](#).

9.4.2.12 char dielZpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to z-shifted dielectric map files

Definition at line [231](#) of file [nosh.h](#).

9.4.2.13 NOsh_calc* elec[NOSH_MAXCALC]

The array of calculation objects corresponding to ELEC statements read in the input file.
Compare to [sNOsh::calc](#)

Definition at line [189](#) of file [nosh.h](#).

9.4.2.14 int elec2calc[NOSH_MAXCALC]

A mapping between ELEC statements which appear in the input file and calc objects stored above. Since we allow both normal and focused multigrid, there isn't a 1-to-1 correspondence between ELEC statements and actual calculations. This can really confuse operations which work on specific calculations further down the road (like PRINT). Therefore this array is the initial point of entry for any calculation-specific operation. It points to a specific entry in the calc array.

Definition at line [208](#) of file [nosh.h](#).

9.4.2.15 char elecname[NOSH_MAXCALC][VMAX_ARGLEN]

Optional user-specified name for ELEC statement

Definition at line [254](#) of file [nosh.h](#).

9.4.2.16 int gotparm

Either have (1) or don't have (0) parm

Definition at line [223](#) of file [nosh.h](#).

9.4.2.17 int ispara

1 => is a parallel calculation, 0 => is not

Definition at line [201](#) of file [nosh.h](#).

9.4.2.18 Vdata_Format kappafmt[NOSH_MAXMOL]

Kappa maps file formats

Definition at line [236](#) of file [nosh.h](#).

9.4.2.19 char kappapath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to kappa map files

Definition at line [235](#) of file [nosh.h](#).

9.4.2.20 Vdata_Format meshfmt[NOSH_MAXMOL]

Mesh fileformats

Definition at line [245](#) of file [nosh.h](#).

9.4.2.21 char meshpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to mesh files

Definition at line [244](#) of file [nosh.h](#).

9.4.2.22 NOsh_MolFormat molfmt[NOSH_MAXMOL]

Mol files formats

Definition at line [220](#) of file [nosh.h](#).

9.4.2.23 char molpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to mol files

Definition at line [219](#) of file [nosh.h](#).

9.4.2.24 int napol

The number of apolar statements in the input file and in the apolar array

Definition at line [198](#) of file [nosh.h](#).

9.4.2.25 int ncalc

The number of calculations in the calc array

Definition at line [187](#) of file [nosh.h](#).

9.4.2.26 int ncharge

Number of charge maps

Definition at line [240](#) of file [nosh.h](#).

9.4.2.27 int ndiel

Number of dielectric maps

Definition at line [226](#) of file [nosh.h](#).

9.4.2.28 int nelec

The number of elec statements in the input file and in the elec array

Definition at line [192](#) of file [nosh.h](#).

9.4.2.29 int nkappa

Number of kappa maps

Definition at line [234](#) of file [nosh.h](#).

9.4.2.30 int nmesh

Number of meshes

Definition at line [243](#) of file [nosh.h](#).

9.4.2.31 int nmol

Number of molecules

Definition at line [218](#) of file [nosh.h](#).

9.4.2.32 int npot

Number of potential maps

Definition at line [237](#) of file [nosh.h](#).

9.4.2.33 int nprint

How many print sections?

Definition at line [246](#) of file [nosh.h](#).

9.4.2.34 NOsh_ParmFormat parmfmt

Parm file format

Definition at line [225](#) of file [nosh.h](#).

9.4.2.35 char parmpath[VMAX_ARGLEN]

Paths to parm file

Definition at line [224](#) of file [nosh.h](#).

9.4.2.36 int parsed

Have we parsed an input file yet?

Definition at line [253](#) of file [nosh.h](#).

9.4.2.37 Vdata_Format potfmt[NOSH_MAXMOL]

Potential maps file formats

Definition at line [239](#) of file [nosh.h](#).

9.4.2.38 char potpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to potential map files

Definition at line [238](#) of file [nosh.h](#).

9.4.2.39 int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP]

ELEC id (see elec2calc)

Definition at line [250](#) of file [nosh.h](#).

9.4.2.40 int printnarg[NOSH_MAXPRINT]

How many arguments in energy list

Definition at line [249](#) of file [nosh.h](#).

9.4.2.41 int printop[NOSH_MAXPRINT][NOSH_MAXPOP]

Operation id (0 = add, 1 = subtract)

Definition at line [251](#) of file [nosh.h](#).

9.4.2.42 NOsh_PrintType printwhat[NOSH_MAXPRINT]

What do we print:

- 0 = energy,
- 1 = force

Definition at line [247](#) of file [nosh.h](#).

9.4.2.43 int proc_rank

Processor rank in parallel calculation

Definition at line 202 of file [nosh.h](#).

9.4.2.44 int proc_size

Number of processors in parallel calculation

Definition at line 203 of file [nosh.h](#).

The documentation for this struct was generated from the following file:

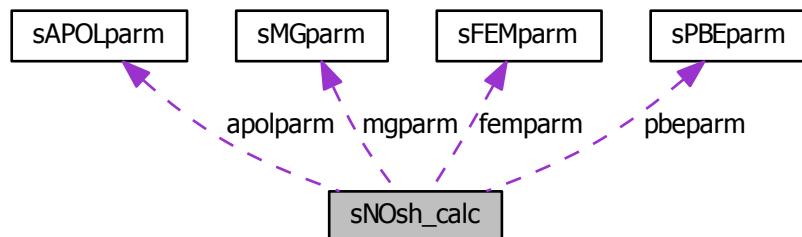
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[nosh.h](#)

9.5 sNOsh_calc Struct Reference

Calculation class for use when parsing fixed format input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/nosh.h>
```

Collaboration diagram for sNOsh_calc:



Data Fields

- [MGparm * mgparm](#)

- FEMparm * femparm
- PBEparm * pbeparm
- APOLparm * apolparm
- NOsh_CalcType calctype

9.5.1 Detailed Description

Calculation class for use when parsing fixed format input files.

Author

Nathan Baker

Definition at line 163 of file [nosh.h](#).

9.5.2 Field Documentation

9.5.2.1 APOLparm* apolparm

Non-polar parameters

Definition at line 167 of file [nosh.h](#).

9.5.2.2 NOsh_CalcType calctype

Calculation type

Definition at line 168 of file [nosh.h](#).

9.5.2.3 FEMparm* femparm

Finite element parameters

Definition at line 165 of file [nosh.h](#).

9.5.2.4 MGparm* mgparm

Multigrid parameters

Definition at line 164 of file [nosh.h](#).

9.5.2.5 PBEparm* pbeparm

Generic PBE parameters

Definition at line 166 of file [nosh.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[nosh.h](#)

9.6 sPBEparm Struct Reference

Parameter structure for PBE variables from input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/pbeparm.h>
```

Data Fields

- int `molid`
- int `setmolid`
- int `useDielMap`
- int `dielMapID`
- int `useKappaMap`
- int `kappaMapID`
- int `usePotMap`
- int `potMapID`
- int `useChargeMap`
- int `chargeMapID`
- `Vhal_PBEType pbetype`
- int `setpbetype`
- `Vbcfl bcfl`
- int `setbcfl`
- int `nion`
- int `setnion`
- double `ionq` [MAXION]
- double `ionc` [MAXION]
- double `ionr` [MAXION]
- int `setion` [MAXION]
- double `pdie`
- int `setpdie`
- double `sdens`

- int `setsdens`
- double `sdie`
- int `setsdie`
- `Vsurf_Meth` `srfm`
- int `setsrfm`
- double `srad`
- int `setsrad`
- double `swin`
- int `setswin`
- double `temp`
- int `settemp`
- double `smsize`
- int `setsmsize`
- double `smvolume`
- int `setsmvolume`
- `PBEparm_calcEnergy` `calcenergy`
- int `setcalcenergy`
- `PBEparm_calcForce` `calcforce`
- int `setcalcforce`
- double `zmem`
- int `setzmem`
- double `Lmem`
- int `setLmem`
- double `mdie`
- int `setmdie`
- double `memv`
- int `setmemv`
- int `numwrite`
- char `writestem` [PBEPARM_MAXWRITE][VMAX_ARGLEN]
- `Vdata_Type` `writetype` [PBEPARM_MAXWRITE]
- `Vdata_Format` `writefmt` [PBEPARM_MAXWRITE]
- int `writemat`
- int `setwritemat`
- char `writematstem` [VMAX_ARGLEN]
- int `writematflag`
- int `parsed`

9.6.1 Detailed Description

Parameter structure for PBE variables from input files.

Author

Nathan Baker

Note

If you add/delete/change something in this class, the member functions -- especially PBEparm_copy -- must be modified accordingly

Definition at line 116 of file [pbeparm.h](#).

9.6.2 Field Documentation

9.6.2.1 Vbcfl bcfl

Boundary condition method

Definition at line 135 of file [pbeparm.h](#).

9.6.2.2 PBEparm_calcEnergy calcenergy

Energy calculation flag

Definition at line 164 of file [pbeparm.h](#).

9.6.2.3 PBEparm_calcForce calcforce

Atomic forces calculation

Definition at line 166 of file [pbeparm.h](#).

9.6.2.4 int chargeMapID

Charge distribution map ID (if used)

Definition at line 132 of file [pbeparm.h](#).

9.6.2.5 int dielMapID

Dielectric map ID (if used)

Definition at line 122 of file [pbeparm.h](#).

9.6.2.6 double ionc[MAXION]

Counterion concentrations (in M)

Definition at line 140 of file [pbparm.h](#).

9.6.2.7 double ionq[MAXION]

Counterion charges (in e)

Definition at line 139 of file [pbparm.h](#).

9.6.2.8 double ionr[MAXION]

Counterion radii (in A)

Definition at line 141 of file [pbparm.h](#).

9.6.2.9 int kappaMapID

Kappa map ID (if used)

Definition at line 125 of file [pbparm.h](#).

9.6.2.10 double Lmem

membrane width

Definition at line 175 of file [pbparm.h](#).

9.6.2.11 double mdie

membrane dielectric constant

Definition at line 177 of file [pbparm.h](#).

9.6.2.12 double memv

Membrane potential

Definition at line 179 of file [pbparm.h](#).

9.6.2.13 int molid

Molecule ID to perform calculation on
Definition at line 118 of file [pbeparm.h](#).

9.6.2.14 int nion

Number of counterion species
Definition at line 137 of file [pbeparm.h](#).

9.6.2.15 int numwrite

Number of write statements encountered
Definition at line 184 of file [pbeparm.h](#).

9.6.2.16 int parsed

Has this been filled with anything other than the default values?
Definition at line 200 of file [pbeparm.h](#).

9.6.2.17 Vhal_PBEType pbetype

Which version of the PBE are we solving?
Definition at line 133 of file [pbeparm.h](#).

9.6.2.18 double pdie

Solute dielectric
Definition at line 143 of file [pbeparm.h](#).

9.6.2.19 int potMapID

Kappa map ID (if used)
Definition at line 128 of file [pbeparm.h](#).

9.6.2.20 double sdens

Vacc sphere density

Definition at line [145](#) of file [pbeparm.h](#).

9.6.2.21 double sdie

Solvent dielectric

Definition at line [147](#) of file [pbeparm.h](#).

9.6.2.22 int setbcfl

Flag,

See also

[bcfl](#)

Definition at line [136](#) of file [pbeparm.h](#).

9.6.2.23 int setcalcenergy

Flag,

See also

[calcenergy](#)

Definition at line [165](#) of file [pbeparm.h](#).

9.6.2.24 int setcalcforce

Flag,

See also

[calcforce](#)

Definition at line [167](#) of file [pbeparm.h](#).

9.6.2.25 int setion[MAXION]

Flag,

See also

[ionq](#)

Definition at line [142](#) of file [pbeparm.h](#).

9.6.2.26 int setLmem

Flag

Definition at line [176](#) of file [pbeparm.h](#).

9.6.2.27 int setmdie

Flag

Definition at line [178](#) of file [pbeparm.h](#).

9.6.2.28 int setmemv

Flag

Definition at line [180](#) of file [pbeparm.h](#).

9.6.2.29 int setmolid

Flag,

See also

[molid](#)

Definition at line [119](#) of file [pbeparm.h](#).

9.6.2.30 int setnion

Flag,

See also

[nion](#)

Definition at line [138](#) of file [pbeparm.h](#).

9.6.2.31 int setpbetype

Flag,

See also

[pbetype](#)

Definition at line [134](#) of file [pbeparm.h](#).

9.6.2.32 int setpdie

Flag,

See also

[pdie](#)

Definition at line [144](#) of file [pbeparm.h](#).

9.6.2.33 int setsdens

Flag,

See also

[sdens](#)

Definition at line [146](#) of file [pbeparm.h](#).

9.6.2.34 int setsdie

Flag,

See also

[sdie](#)

Definition at line [148](#) of file [pbeparm.h](#).

9.6.2.35 int setsmsize

Flag,

See also[temp](#)

Definition at line 159 of file [pbeparm.h](#).

9.6.2.36 int setsvolume

Flag,

See also[temp](#)

Definition at line 162 of file [pbeparm.h](#).

9.6.2.37 int setsrad

Flag,

See also[srad](#)

Definition at line 152 of file [pbeparm.h](#).

9.6.2.38 int setsrfm

Flag,

See also[srfm](#)

Definition at line 150 of file [pbeparm.h](#).

9.6.2.39 int setswin

Flag,

See also[swin](#)

Definition at line 154 of file [pbeparm.h](#).

9.6.2.40 int settemp

Flag,

See also

[temp](#)

Definition at line 156 of file pbeparm.h.

9.6.2.41 int setwritemat

Flag,

See also

[writemat](#)

Definition at line 193 of file pbeparm.h.

9.6.2.42 int setzmem

Flag

Definition at line 174 of file pbeparm.h.

9.6.2.43 double smsize

SMPBE size

Definition at line 158 of file pbeparm.h.

9.6.2.44 double smvolume

SMPBE size

Definition at line 161 of file pbeparm.h.

9.6.2.45 double srad

Solvent radius

Definition at line 151 of file pbeparm.h.

9.6.2.46 Vsurf_Meth srfm

Surface calculation method

Definition at line 149 of file [pbeparm.h](#).

9.6.2.47 double swin

Cubic spline window

Definition at line 153 of file [pbeparm.h](#).

9.6.2.48 double temp

Temperature (in K)

Definition at line 155 of file [pbeparm.h](#).

9.6.2.49 int useChargeMap

Indicates whether we use an external charge distribution map

Definition at line 130 of file [pbeparm.h](#).

9.6.2.50 int useDielMap

Indicates whether we use external dielectric maps (note plural)

Definition at line 120 of file [pbeparm.h](#).

9.6.2.51 int useKappaMap

Indicates whether we use an external kappa map

Definition at line 123 of file [pbeparm.h](#).

9.6.2.52 int usePotMap

Indicates whether we use an external kappa map

Definition at line 126 of file [pbeparm.h](#).

9.6.2.53 Vdata_Format writefmt[PBEPARM_MAXWRITE]

File format to write data in

Definition at line 188 of file pbeparm.h.

9.6.2.54 int writemat

Write out the operator matrix?

- 0 => no
- 1 => yes

Definition at line 190 of file pbeparm.h.

9.6.2.55 int writematflag

What matrix should we write:

- 0 => Poisson (differential operator)
- 1 => Poisson-Boltzmann operator linearized around solution (if applicable)

Definition at line 195 of file pbeparm.h.

9.6.2.56 char writematstem[VMAX_ARGLEN]

File stem to write mat

Definition at line 194 of file pbeparm.h.

9.6.2.57 char writestem[PBEPARM_MAXWRITE][VMAX_ARGLEN]

File stem to write data to

Definition at line 185 of file pbeparm.h.

9.6.2.58 Vdata_Type writetype[PBEPARM_MAXWRITE]

What data to write

Definition at line 187 of file pbeparm.h.

9.6.2.59 double zmem

z value of membrane bottom

Definition at line 173 of file [pbeparm.h](#).

The documentation for this struct was generated from the following file:

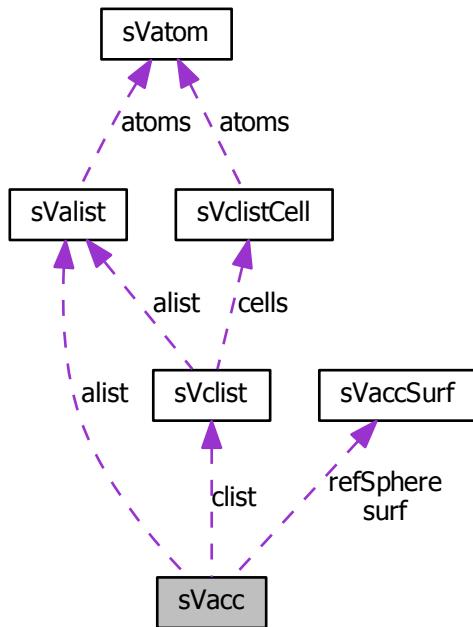
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[pbeparm.h](#)

9.7 sVacc Struct Reference

Oracle for solvent- and ion-accessibility around a biomolecule.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/vacc.h>
```

Collaboration diagram for sVacc:



Data Fields

- Vmem * `mem`
- Valist * `alist`
- Vclist * `clist`
- int * `atomFlags`
- VaccSurf * `refSphere`
- VaccSurf ** `surf`
- Vset `acc`
- double `surf_density`

9.7.1 Detailed Description

Oracle for solvent- and ion-accessibility around a biomolecule.

Author

Nathan Baker

Definition at line [105](#) of file [vacc.h](#).

9.7.2 Field Documentation

9.7.2.1 Vset acc

An integer array (to be treated as bitfields) of Vset type with length equal to the number of vertices in the mesh

Definition at line [117](#) of file [vacc.h](#).

9.7.2.2 Valist* alist

Valist structure for list of atoms

Definition at line [108](#) of file [vacc.h](#).

9.7.2.3 int* atomFlags

Array of boolean flags of length Valist_getNumberAtoms(thee->alist) to prevent double-counting atoms during calculations

Definition at line [110](#) of file [vacc.h](#).

9.7.2.4 Vclist* clist

Vclist structure for atom cell list

Definition at line [109](#) of file [vacc.h](#).

9.7.2.5 Vmem* mem

Memory management object for this class

Definition at line [107](#) of file [vacc.h](#).

9.7.2.6 VaccSurf* refSphere

Reference sphere for SASA calculations

Definition at line [113](#) of file [vacc.h](#).

9.7.2.7 VaccSurf** surf

Array of surface points for each atom; is not initialized until needed (test against VNULL to determine initialization state)

Definition at line 114 of file [vacc.h](#).

9.7.2.8 double surf_density

Minimum solvent accessible surface point density (in pts/A²)

Definition at line 119 of file [vacc.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[vacc.h](#)

9.8 sVaccSurf Struct Reference

Surface object list of per-atom surface points.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/vacc.h>
```

Data Fields

- Vmem * [mem](#)
- double * [xpts](#)
- double * [ypts](#)
- double * [zpts](#)
- char * [bpts](#)
- double [area](#)
- int [npts](#)
- double [probe_radius](#)

9.8.1 Detailed Description

Surface object list of per-atom surface points.

Author

Nathan Baker

Definition at line 81 of file [vacc.h](#).

9.8.2 Field Documentation

9.8.2.1 double area

Area spanned by these points

Definition at line [88](#) of file [vacc.h](#).

9.8.2.2 char* bpts

Array of booleans indicating whether a point is (1) or is not (0) part of the surface

Definition at line [86](#) of file [vacc.h](#).

9.8.2.3 Vmem* mem

Memory object

Definition at line [82](#) of file [vacc.h](#).

9.8.2.4 int npts

Length of thee->xpts, ypts, zpts arrays

Definition at line [89](#) of file [vacc.h](#).

9.8.2.5 double probe_radius

Probe radius (A) with which this surface was constructed

Definition at line [90](#) of file [vacc.h](#).

9.8.2.6 double* xpts

Array of point x-locations

Definition at line [83](#) of file [vacc.h](#).

9.8.2.7 double* ypts

Array of point y-locations

Definition at line [84](#) of file [vacc.h](#).

9.8.2.8 double* zpts

Array of point z-locations

Definition at line 85 of file [vacc.h](#).

The documentation for this struct was generated from the following file:

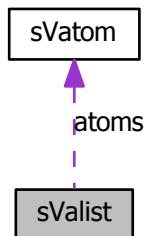
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[vacc.h](#)

9.9 sValist Struct Reference

Container class for list of atom objects.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/valist.h>
```

Collaboration diagram for sValist:



Data Fields

- int [number](#)
- double [center](#) [3]
- double [mincrd](#) [3]
- double [maxcrd](#) [3]
- double [maxrad](#)
- double [charge](#)
- [Vatom](#) * [atoms](#)

- Vmem * **vmem**

9.9.1 Detailed Description

Container class for list of atom objects.

Author

Nathan Baker

Definition at line [78](#) of file [valist.h](#).

9.9.2 Field Documentation

9.9.2.1 **Vatom*** atoms

Atom list

Definition at line [86](#) of file [valist.h](#).

9.9.2.2 double center[3]

Molecule center (xmin - xmax)/2, etc.

Definition at line [81](#) of file [valist.h](#).

9.9.2.3 double charge

Net charge

Definition at line [85](#) of file [valist.h](#).

9.9.2.4 double maxcrd[3]

Maximum coordinates

Definition at line [83](#) of file [valist.h](#).

9.9.2.5 double maxrad

Maximum radius

Definition at line [84](#) of file [valist.h](#).

9.9.2.6 double mincrd[3]

Minimum coordinates

Definition at line 82 of file [valist.h](#).

9.9.2.7 int number

Number of atoms in list

Definition at line 80 of file [valist.h](#).

9.9.2.8 Vmem* vmem

Memory management object

Definition at line 87 of file [valist.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[valist.h](#)

9.10 sVatom Struct Reference

Contains public data members for Vatom class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/vatom.h>
```

Data Fields

- double [position](#) [3]
- double [radius](#)
- double [charge](#)
- double [partID](#)
- double [epsilon](#)
- int [id](#)
- char [resName](#) [VMAX_RECLEN]
- char [atomName](#) [VMAX_RECLEN]

9.10.1 Detailed Description

Contains public data members for Vatom class/module.

Author

Nathan Baker, David Gohara, Mike Schneiders

Definition at line 81 of file [vatom.h](#).

9.10.2 Field Documentation

9.10.2.1 char atomName[VMAX_RECLEN]

Atom name from PDB/PDR file

Definition at line 95 of file [vatom.h](#).

9.10.2.2 double charge

Atomic charge

Definition at line 85 of file [vatom.h](#).

9.10.2.3 double epsilon

Epsilon value for WCA calculations

Definition at line 88 of file [vatom.h](#).

9.10.2.4 int id

Atomic ID; this should be a unique non-negative integer assigned based on the index of the atom in a Valist atom array

Definition at line 90 of file [vatom.h](#).

9.10.2.5 double partID

Partition value for assigning atoms to particular processors and/or partitions

Definition at line 86 of file [vatom.h](#).

9.10.2.6 double position[3]

Atomic position

Definition at line 83 of file [vatom.h](#).

9.10.2.7 double radius

Atomic radius

Definition at line 84 of file [vatom.h](#).

9.10.2.8 char resName[VMAX_RECLEN]

Residue name from PDB/PQR file

Definition at line 94 of file [vatom.h](#).

The documentation for this struct was generated from the following file:

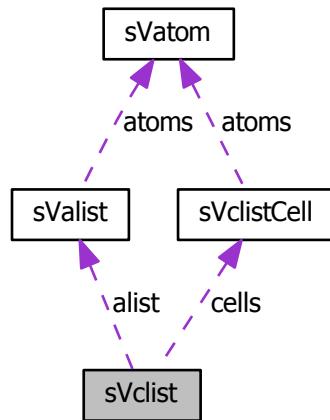
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[vatom.h](#)

9.11 sVclist Struct Reference

Atom cell list.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/vclist.h>
```

Collaboration diagram for sVclist:



Data Fields

- Vmem * `vmem`
- Valist * `alist`
- Vclist_DomainMode `mode`
- int `npts` [VAPBS_DIM]
- int `n`
- double `max_radius`
- VclistCell * `cells`
- double `lower_corner` [VAPBS_DIM]
- double `upper_corner` [VAPBS_DIM]
- double `spacs` [VAPBS_DIM]

9.11.1 Detailed Description

Atom cell list.

Author

Nathan Baker

Definition at line 114 of file [vclist.h](#).

9.11.2 Field Documentation

9.11.2.1 Valist* **alist**

Original Valist structure for list of atoms

Definition at line 117 of file [vclist.h](#).

9.11.2.2 VclistCell* **cells**

Cell array of length thee->n

Definition at line 122 of file [vclist.h](#).

9.11.2.3 double **lower_corner[VAPBS_DIM]**

Hash table grid corner

Definition at line 123 of file [vclist.h](#).

9.11.2.4 double **max_radius**

Maximum probe radius

Definition at line 121 of file [vclist.h](#).

9.11.2.5 Vclist_DomainMode **mode**

How the cell list was constructed

Definition at line 118 of file [vclist.h](#).

9.11.2.6 int **n**

$n = nx \times nz \times ny$

Definition at line 120 of file [vclist.h](#).

9.11.2.7 int **npts[VAPBS_DIM]**

Hash table grid dimensions

Definition at line 119 of file [vcclist.h](#).

9.11.2.8 double **spacs**[VAPBS_DIM]

Hash table grid spacings

Definition at line 125 of file [vcclist.h](#).

9.11.2.9 double **upper_corner**[VAPBS_DIM]

Hash table grid corner

Definition at line 124 of file [vcclist.h](#).

9.11.2.10 Vmem* **vmem**

Memory management object for this class

Definition at line 116 of file [vcclist.h](#).

The documentation for this struct was generated from the following file:

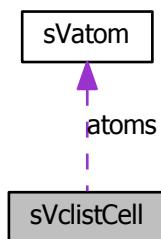
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[vcclist.h](#)

9.12 sVclistCell Struct Reference

Atom cell list cell.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vclist.h>
```

Collaboration diagram for sVclistCell:



Data Fields

- `Vatom ** atoms`
- int `natoms`

9.12.1 Detailed Description

Atom cell list cell.

Author

Nathan Baker

Definition at line [98](#) of file [vclist.h](#).

9.12.2 Field Documentation

9.12.2.1 Vatom** atoms

Array of atom objects associated with this cell

Definition at line [99](#) of file [vclist.h](#).

9.12.2.2 int natoms

Length of thee->atoms array

Definition at line 100 of file [vclist.h](#).

The documentation for this struct was generated from the following file:

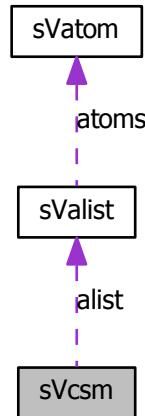
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[vclist.h](#)

9.13 sVcsm Struct Reference

Charge-simplex map class.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source
code/APBS/src/fem/apbs/vcsm.h>
```

Collaboration diagram for sVcsm:



Data Fields

- [Valist * alist](#)
- int [natom](#)
- Gem * [gm](#)
- int ** [sqm](#)
- int * [nsqm](#)

- int **nsimp**
- int **msimp**
- int ** **qsm**
- int * **nqsm**
- int **initFlag**
- Vmem * **vmem**

9.13.1 Detailed Description

Charge-simplex map class.

Author

Nathan Baker

Definition at line [89](#) of file [vcsm.h](#).

9.13.2 Field Documentation

9.13.2.1 Valist* **alist**

Atom (charge) list

Definition at line [91](#) of file [vcsm.h](#).

9.13.2.2 Gem* **gm**

Grid manager (container class for master vertex and simplex lists as well as prolongation operator for updating after refinement)

Definition at line [94](#) of file [vcsm.h](#).

9.13.2.3 int **initFlag**

Indicates whether the maps have been initialized yet

Definition at line [112](#) of file [vcsm.h](#).

9.13.2.4 int **msimp**

The maximum number of entries that can be accomodated by sqm or nsqm -- saves on realloc's

Definition at line [107](#) of file [vcsm.h](#).

9.13.2.5 int natom

Size of thee->alist; redundant, but useful for convenience

Definition at line [92](#) of file [vcsm.h](#).

9.13.2.6 int* nqsm

The length of the simplex lists in thee->qsm

Definition at line [111](#) of file [vcsm.h](#).

9.13.2.7 int nsimp

The _currently used) length of sqm, nsqm -- may not always be up-to-date with Gem

Definition at line [105](#) of file [vcsm.h](#).

9.13.2.8 int* nsqm

The length of the charge lists in thee->sqm

Definition at line [104](#) of file [vcsm.h](#).

9.13.2.9 int qsm**

The inverse of sqm; the list of simplices associated with a given charge

Definition at line [109](#) of file [vcsm.h](#).

9.13.2.10 int sqm**

The map which gives the list charges associated with each simplex in gm->simplices.
The indices of the first dimension are associated with the simplex ID's in Vgm. Each
charge list (second dimension) contains entries corresponding to indicies in thee->alist
with lengths given in thee->nsmq

Definition at line [97](#) of file [vcsm.h](#).

9.13.2.11 Vmem* vmem

Memory management object

Definition at line [114](#) of file [vcsm.h](#).

The documentation for this struct was generated from the following file:

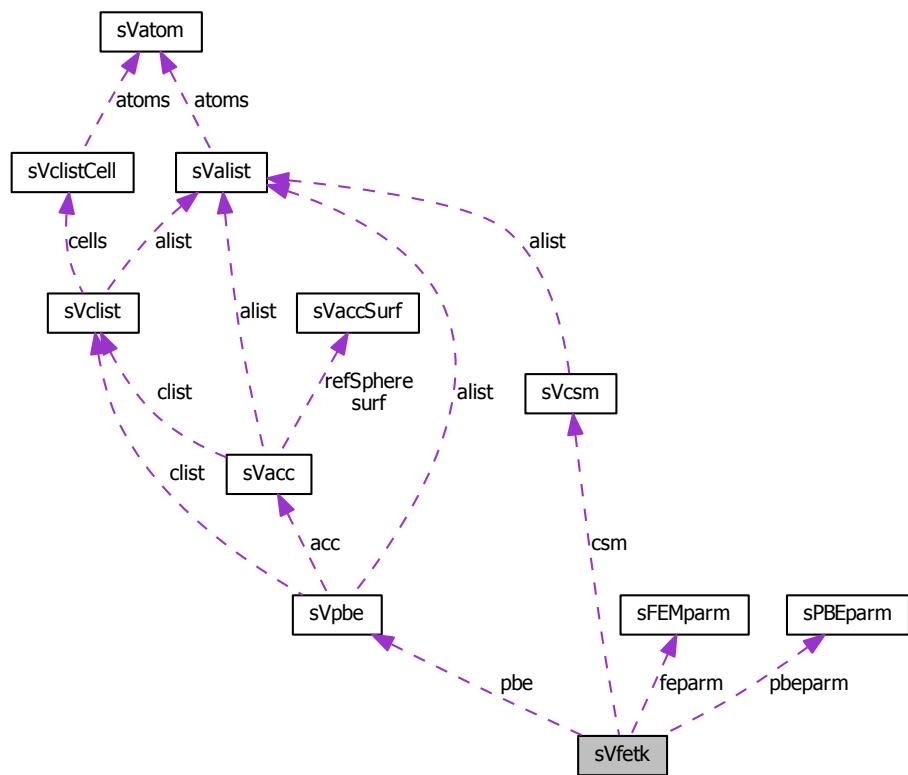
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/[vcsms.h](#)

9.14 sVfetk Struct Reference

Contains public data members for Vfetk class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source
code/APBS/src/fem/apbs/vfetk.h>
```

Collaboration diagram for sVfetk:



Data Fields

- Vmem * [vmem](#)
- Gem * [gm](#)
- AM * [am](#)
- Aprx * [aprx](#)
- PDE * [pde](#)
- Vpbe * [pbe](#)
- Vcsm * [csm](#)
- Vfetk_LsolvType [lkey](#)
- int [lmax](#)
- double [ltol](#)
- Vfetk_NsolvType [nkey](#)
- int [nmax](#)
- double [ntol](#)
- Vfetk_GuessType [gues](#)
- Vfetk_PrecType [lprec](#)
- int [pjac](#)
- PBEparm * [pbeparm](#)
- FEMparm * [feparm](#)
- Vhal_PBEType [type](#)
- int [level](#)

9.14.1 Detailed Description

Contains public data members for Vfetk class/module.

Author

Nathan Baker Many of the routines and macros are borrowed from the main.c driver (written by Mike Holst) provided with the PMG code.

Definition at line 173 of file [vfetk.h](#).

9.14.2 Field Documentation

9.14.2.1 AM* am

Multilevel algebra manager.

Definition at line 179 of file [vfetk.h](#).

9.14.2.2 Aprx* aprx

Approximation manager.

Definition at line 180 of file [vfetk.h](#).

9.14.2.3 Vcsm* csm

Charge-simplex map

Definition at line 183 of file [vfetk.h](#).

9.14.2.4 FEMparm* feparm

FEM-specific parameters

Definition at line 195 of file [vfetk.h](#).

9.14.2.5 Gem* gm

Grid manager (container class for master vertex and simplex lists as well as prolongation operator for updating after refinement).

Definition at line 176 of file [vfetk.h](#).

9.14.2.6 Vfetk_GuessType gues

Initial guess method

Definition at line 190 of file [vfetk.h](#).

9.14.2.7 int level

Refinement level (starts at 0)

Definition at line 197 of file [vfetk.h](#).

9.14.2.8 Vfetk_LsolvType lkey

Linear solver method

Definition at line 184 of file [vfetk.h](#).

9.14.2.9 int lmax

Maximum number of linear solver iterations

Definition at line 185 of file [vfetk.h](#).

9.14.2.10 Vfetk_PrecType lprec

Linear preconditioner

Definition at line 191 of file [vfetk.h](#).

9.14.2.11 double ltol

Residual tolerance for linear solver

Definition at line 186 of file [vfetk.h](#).

9.14.2.12 Vfetk_NsolvType nkey

Nonlinear solver method

Definition at line 187 of file [vfetk.h](#).

9.14.2.13 int nmax

Maximum number of nonlinear solver iterations

Definition at line 188 of file [vfetk.h](#).

9.14.2.14 double ntol

Residual tolerance for nonlinear solver

Definition at line 189 of file [vfetk.h](#).

9.14.2.15 Vpbe* pbe

Poisson-Boltzmann object

Definition at line 182 of file [vfetk.h](#).

9.14.2.16 PBEparm* pbeparm

Generic PB parameters

Definition at line 194 of file [vfetk.h](#).

9.14.2.17 PDE* pde

FEtk PDE object

Definition at line 181 of file [vfetk.h](#).

9.14.2.18 int pjac

Flag to print the jacobians (usually set this to -1, please)

Definition at line 192 of file [vfetk.h](#).

9.14.2.19 Vhal_PBEType type

Version of PBE to solve

Definition at line 196 of file [vfetk.h](#).

9.14.2.20 Vmem* vmem

Memory management object

Definition at line 175 of file [vfetk.h](#).

The documentation for this struct was generated from the following file:

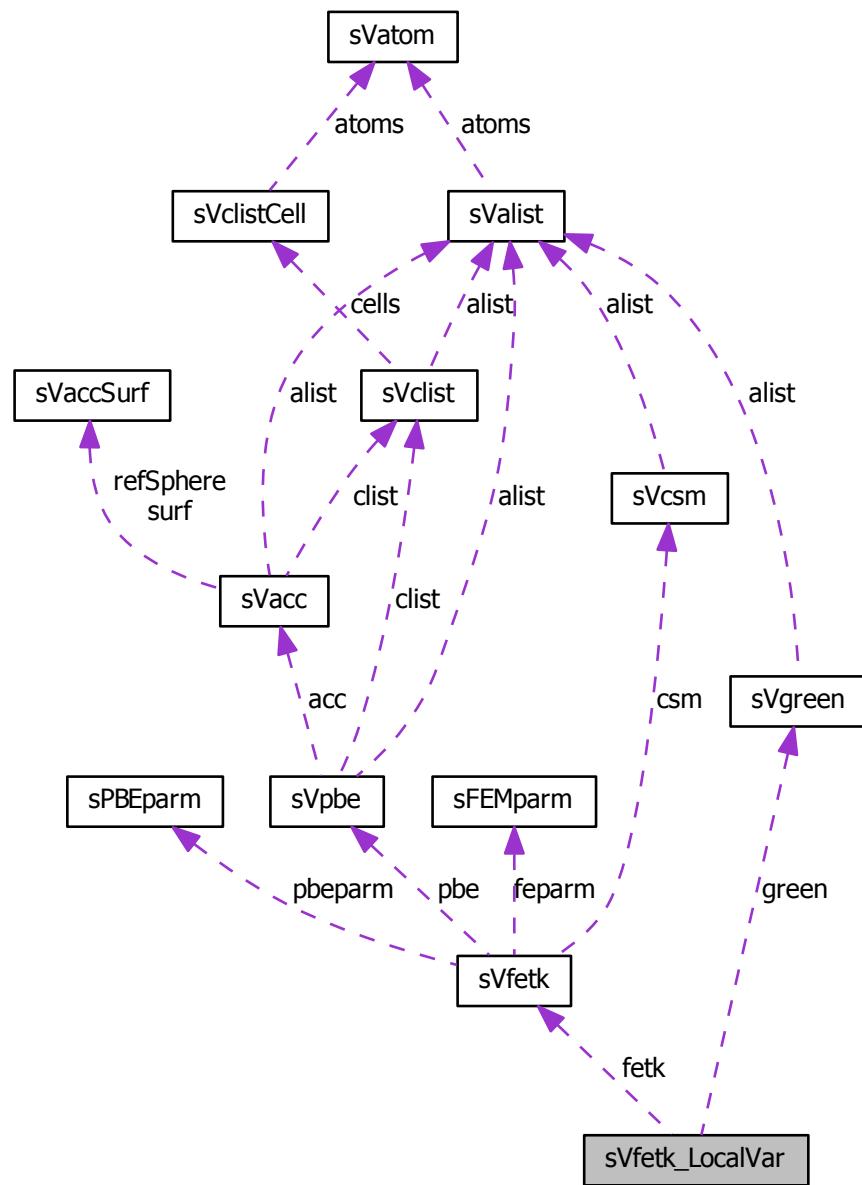
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/[vfetk.h](#)

9.15 sVfetk_LocalVar Struct Reference

Vfetk LocalVar subclass.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/vfetk.h>
```

Collaboration diagram for sVfetk_LocalVar:



Data Fields

- double `nvec` [VAPBS_DIM]
- double `vx` [4][VAPBS_DIM]
- double `xq` [VAPBS_DIM]
- double `U` [MAXV]
- double `dU` [MAXV][VAPBS_DIM]
- double `W`
- double `dW` [VAPBS_DIM]
- double `d2W`
- int `sType`
- int `fType`
- double `diel`
- double `ionacc`
- double `A`
- double `F`
- double `B`
- double `DB`
- double `jumpDiel`
- `Vfetk` * `fetk`
- `Vgreen` * `green`
- int `initGreen`
- `SS` * `simp`
- `VV` * `verts` [4]
- int `nverts`
- double `ionConc` [MAXION]
- double `ionQ` [MAXION]
- double `ionRadii` [MAXION]
- double `zkappa2`
- double `zks2`
- double `ionstr`
- int `nion`
- double `Fu_v`
- double `DFu_wv`
- double `delta`
- double `u_D`
- double `u_T`

9.15.1 Detailed Description

Vfetk LocalVar subclass.

Author

Nathan Baker Contains variables used when solving the PDE with FEtk

Definition at line [212](#) of file [vfetk.h](#).

9.15.2 Field Documentation

9.15.2.1 double A

Second-order differential term

Definition at line [225](#) of file [vfetk.h](#).

9.15.2.2 double B

Entire ionic strength term

Definition at line [227](#) of file [vfetk.h](#).

9.15.2.3 double d2W

Coulomb regularization term Laplacian

Definition at line [220](#) of file [vfetk.h](#).

9.15.2.4 double DB

Entire ionic strength term derivative

Definition at line [228](#) of file [vfetk.h](#).

9.15.2.5 double delta

Store delta value

Definition at line [247](#) of file [vfetk.h](#).

9.15.2.6 double DFu_wv

Store DFu_wv value

Definition at line [246](#) of file [vfetk.h](#).

9.15.2.7 double diel

Dielectric value

Definition at line [223](#) of file [vfetk.h](#).

9.15.2.8 double dU[MAXV][VAPBS_DIM]

Solution gradient

Definition at line [217](#) of file [vfetk.h](#).

9.15.2.9 double dW[VAPBS_DIM]

Coulomb regularization term gradient

Definition at line [219](#) of file [vfetk.h](#).

9.15.2.10 double F

RHS characteristic function value

Definition at line [226](#) of file [vfetk.h](#).

9.15.2.11 Vfetk* fetk

Pointer to the VFETK object

Definition at line [230](#) of file [vfetk.h](#).

9.15.2.12 int fType

Face type

Definition at line [222](#) of file [vfetk.h](#).

9.15.2.13 double Fu_v

Store Fu_v value

Definition at line [245](#) of file [vfetk.h](#).

9.15.2.14 Vgreen* green

Pointer to a Green's function object

Definition at line [231](#) of file [vfetk.h](#).

9.15.2.15 int initGreen

Boolean to designate whether Green's function has been initialized

Definition at line [232](#) of file [vfetk.h](#).

9.15.2.16 double ionacc

Ion accessibility value

Definition at line [224](#) of file [vfetk.h](#).

9.15.2.17 double ionConc[MAXION]

Counterion species' concentrations

Definition at line [238](#) of file [vfetk.h](#).

9.15.2.18 double ionQ[MAXION]

Counterion species' valencies

Definition at line [239](#) of file [vfetk.h](#).

9.15.2.19 double ionRadii[MAXION]

Counterion species' radii

Definition at line [240](#) of file [vfetk.h](#).

9.15.2.20 double ionstr

Ionic strength parameters (M)

Definition at line [243](#) of file [vfetk.h](#).

9.15.2.21 double jumpDiel

Dielectric value on one side of a simplex face

Definition at line [229](#) of file [vfetk.h](#).

9.15.2.22 int nion

Number of ion species

Definition at line [244](#) of file [vfetk.h](#).

9.15.2.23 double nvec[VAPBS_DIM]

Normal vector for a simplex face

Definition at line [213](#) of file [vfetk.h](#).

9.15.2.24 int nverts

number of vertices in the simplex

Definition at line [237](#) of file [vfetk.h](#).

9.15.2.25 SS* simp

Pointer to the latest simplex object; set in initElement() and [delta\(\)](#)

Definition at line [234](#) of file [vfetk.h](#).

9.15.2.26 int sType

Simplex type

Definition at line [221](#) of file [vfetk.h](#).

9.15.2.27 double U[MAXV]

Solution value

Definition at line [216](#) of file [vfetk.h](#).

9.15.2.28 double u_D

Store Dirichlet value

Definition at line [248](#) of file [vfetk.h](#).

9.15.2.29 double u_T

Store true value

Definition at line [249](#) of file [vfetk.h](#).

9.15.2.30 VV* verts[4]

Pointer to the latest vertices; set in initElement

Definition at line [236](#) of file [vfetk.h](#).

9.15.2.31 double vx[4][VAPBS_DIM]

Vertex coordinates

Definition at line [214](#) of file [vfetk.h](#).

9.15.2.32 double W

Coulomb regularization term scalar value

Definition at line [218](#) of file [vfetk.h](#).

9.15.2.33 double xq[VAPBS_DIM]

Quadrature pt

Definition at line [215](#) of file [vfetk.h](#).

9.15.2.34 double zkappa2

Ionic strength parameters

Definition at line [241](#) of file [vfetk.h](#).

9.15.2.35 double zks2

Ionic strength parameters

Definition at line [242](#) of file [vfetk.h](#).

The documentation for this struct was generated from the following file:

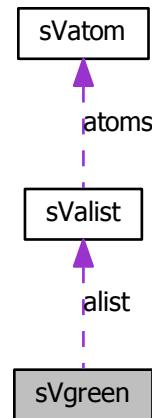
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/[vfetk.h](#)

9.16 sVgreen Struct Reference

Contains public data members for Vgreen class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/vgreen.h>
```

Collaboration diagram for sVgreen:



Data Fields

- `Valist * alist`
- `Vmem * vmem`
- `double * xp`
- `double * yp`
- `double * zp`
- `double * qp`
- `int np`

9.16.1 Detailed Description

Contains public data members for Vgreen class/module.

Author

Nathan Baker

Definition at line 83 of file [vgreen.h](#).

9.16.2 Field Documentation

9.16.2.1 Valist* alist

Atom (charge) list for Green's function

Definition at line 85 of file [vgreen.h](#).

9.16.2.2 int np

Set to size of above arrays

Definition at line 95 of file [vgreen.h](#).

9.16.2.3 double* qp

Array of particle charges for use with treecode routines

Definition at line 93 of file [vgreen.h](#).

9.16.2.4 Vmem* vmem

Memory management object

Definition at line 86 of file [vgreen.h](#).

9.16.2.5 double* xp

Array of particle x-coordinates for use with treecode routines

Definition at line 87 of file [vgreen.h](#).

9.16.2.6 double* yp

Array of particle y-coordinates for use with treecode routines

Definition at line 89 of file [vgreen.h](#).

9.16.2.7 double* zp

Array of particle z-coordinates for use with treecode routines

Definition at line 91 of file [vgreen.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vgreen.h

9.17 sVgrid Struct Reference

Electrostatic potential oracle for Cartesian mesh data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vgrid.h>
```

Data Fields

- int `nx`
- int `ny`
- int `nz`
- double `hx`
- double `hy`
- double `hzed`
- double `xmin`
- double `ymin`
- double `zmin`
- double `xmax`
- double `ymax`
- double `zmax`
- double * `data`
- int `readdata`
- int `ctordata`
- Vmem * `mem`

9.17.1 Detailed Description

Electrostatic potential oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line 79 of file [vgrid.h](#).

9.17.2 Field Documentation

9.17.2.1 int **ctordata**

flag indicating whether data was included at construction

Definition at line 95 of file [vgrid.h](#).

9.17.2.2 double* **data**

$nx*ny*nz$ array of data

Definition at line 93 of file [vgrid.h](#).

9.17.2.3 double **hx**

Grid spacing in x direction

Definition at line 84 of file [vgrid.h](#).

9.17.2.4 double **hy**

Grid spacing in y direction

Definition at line 85 of file [vgrid.h](#).

9.17.2.5 double **hzed**

Grid spacing in z direction

Definition at line 86 of file [vgrid.h](#).

9.17.2.6 Vmem* **mem**

Memory manager object

Definition at line 97 of file [vgrid.h](#).

9.17.2.7 int **nx**

Number grid points in x direction

Definition at line 81 of file [vgrid.h](#).

9.17.2.8 int ny

Number grid points in y direction

Definition at line [82](#) of file [vgrid.h](#).

9.17.2.9 int nz

Number grid points in z direction

Definition at line [83](#) of file [vgrid.h](#).

9.17.2.10 int readdata

flag indicating whether data was read from file

Definition at line [94](#) of file [vgrid.h](#).

9.17.2.11 double xmax

x coordinate of upper grid corner

Definition at line [90](#) of file [vgrid.h](#).

9.17.2.12 double xmin

x coordinate of lower grid corner

Definition at line [87](#) of file [vgrid.h](#).

9.17.2.13 double ymax

y coordinate of upper grid corner

Definition at line [91](#) of file [vgrid.h](#).

9.17.2.14 double ymin

y coordinate of lower grid corner

Definition at line [88](#) of file [vgrid.h](#).

9.17.2.15 double zmax

z coordinate of upper grid corner

Definition at line 92 of file [vgrid.h](#).

9.17.2.16 double zmin

z coordinate of lower grid corner

Definition at line 89 of file [vgrid.h](#).

The documentation for this struct was generated from the following file:

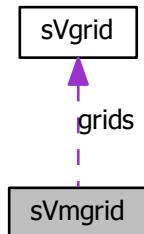
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/[vgrid.h](#)

9.18 sVmgrid Struct Reference

Multiresoltion oracle for Cartesian mesh data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/mg/apbs/vmgrid.h>
```

Collaboration diagram for sVmgrid:



Data Fields

- int [ngrids](#)

- [Vgrid * grids \[VMGRIDMAX\]](#)

9.18.1 Detailed Description

Multiresolution oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line [84](#) of file [vmgrid.h](#).

9.18.2 Field Documentation

9.18.2.1 Vgrid* grids[VMGRIDMAX]

Grids in hierarchy. Our convention will be to have the finest grid first, however, this will not be enforced as it may be useful to search multiple grids for parallel datasets, etc.

Definition at line [87](#) of file [vmgrid.h](#).

9.18.2.2 int ngrids

Number of grids in hierarchy

Definition at line [86](#) of file [vmgrid.h](#).

The documentation for this struct was generated from the following file:

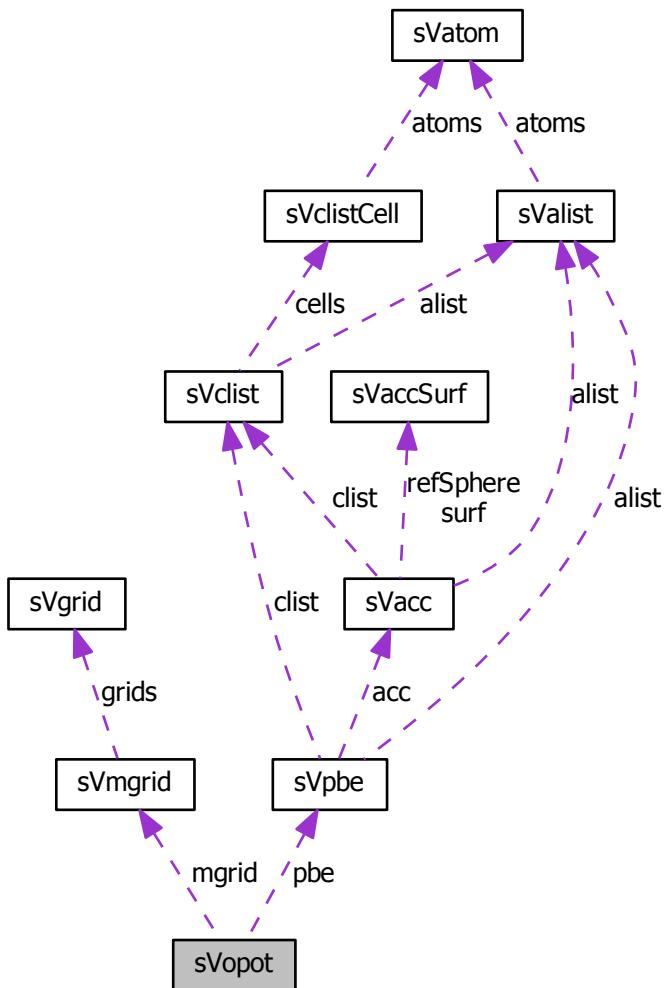
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/[vmgrid.h](#)

9.19 sVopot Struct Reference

Electrostatic potential oracle for Cartesian mesh data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vopot.h>
```

Collaboration diagram for sVopot:



Data Fields

- `Vmgrid * mgrid`

- [Vpbe * pbe](#)
- [Vbcfl bcfl](#)

9.19.1 Detailed Description

Electrostatic potential oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line [82](#) of file [vopot.h](#).

9.19.2 Field Documentation

9.19.2.1 Vbcfl bcfl

Boundary condition flag for returning potential values at points off the grid.

Definition at line [87](#) of file [vopot.h](#).

9.19.2.2 Vmgrid* mgrid

Multiple grid object containing potential data (in units kT/e)

Definition at line [84](#) of file [vopot.h](#).

9.19.2.3 Vpbe* pbe

Pointer to PBE object

Definition at line [86](#) of file [vopot.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/[vopot.h](#)

9.20 sVparam_AtomData Struct Reference

AtomData sub-class; stores atom data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vparam.h>
```

Data Fields

- char `atomName` [VMAX_ARGLEN]
- char `resName` [VMAX_ARGLEN]
- double `charge`
- double `radius`
- double `epsilon`

9.20.1 Detailed Description

AtomData sub-class; stores atom data.

Author

Nathan Baker

Note

The epsilon and radius members of this class refer use the following formula for calculating the van der Waals energy of atom i interacting with atom j :

$$V_{ij}(r_{ij}) = \epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma_{ij}}{r_{ij}} \right)^6 \right]$$

where $\epsilon_{ij} = \sqrt{\epsilon_i \epsilon_j}$ is the well-depth (in the desired energy units), r_{ij} is the distance between atoms i and j , and $\sigma_{ij} = \sigma_i + \sigma_j$ is the sum of the van der Waals radii.

Definition at line 87 of file `vparam.h`.

9.20.2 Field Documentation

9.20.2.1 char `atomName`[VMAX_ARGLEN]

Atom name

Definition at line 88 of file `vparam.h`.

9.20.2.2 double `charge`

Atom charge (in e)

Definition at line 90 of file `vparam.h`.

9.20.2.3 double epsilon

Atom VdW well depth (ε_i above; in kJ/mol)

Definition at line 92 of file [vparam.h](#).

9.20.2.4 double radius

Atom VdW radius (σ_i above; in Å)

Definition at line 91 of file [vparam.h](#).

9.20.2.5 char resName[VMAX_ARGLEN]

Residue name

Definition at line 89 of file [vparam.h](#).

The documentation for this struct was generated from the following file:

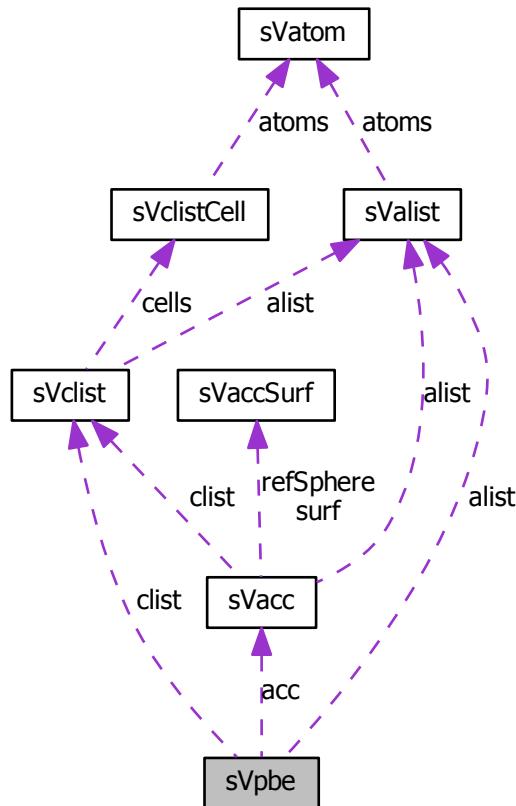
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[vparam.h](#)

9.21 sVpbe Struct Reference

Contains public data members for Vpbe class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/vpbe.h>
```

Collaboration diagram for sVpbe:



Data Fields

- Vmem * vmem
- Valist * alist
- Vclist * clist
- Vacc * acc
- double T
- double soluteDiel
- double solventDiel

- double solventRadius
- double bulkIonicStrength
- double maxIonRadius
- int numIon
- double ionConc [MAXION]
- double ionRadii [MAXION]
- double ionQ [MAXION]
- double xkappa
- double deblen
- double zkappa2
- double zmagic
- double soluteCenter [3]
- double soluteRadius
- double soluteXlen
- double soluteYlen
- double soluteZlen
- double soluteCharge
- double smvolume
- double smsize
- int ipkey
- int paramFlag
- double z_mem
- double L
- double membraneDiel
- double V
- int param2Flag

9.21.1 Detailed Description

Contains public data members for Vpbe class/module.

Author

Nathan Baker

Definition at line 84 of file [vpbe.h](#).

9.21.2 Field Documentation

9.21.2.1 Vacc* acc

Accessibility object

Definition at line 90 of file [vpbe.h](#).

9.21.2.2 Valist* alist

Atom (charge) list

Definition at line [88](#) of file [vpbe.h](#).

9.21.2.3 double bulkIonicStrength

Bulk ionic strength (M)

Definition at line [99](#) of file [vpbe.h](#).

9.21.2.4 Vclist* clist

Atom location cell list

Definition at line [89](#) of file [vpbe.h](#).

9.21.2.5 double debLen

Debye length (bulk)

Definition at line [109](#) of file [vpbe.h](#).

9.21.2.6 double ionConc[MAXION]

Concentration (M) of each species

Definition at line [104](#) of file [vpbe.h](#).

9.21.2.7 double ionQ[MAXION]

Charge (e) of each species

Definition at line [106](#) of file [vpbe.h](#).

9.21.2.8 double ionRadii[MAXION]

Ionic radius (A) of each species

Definition at line [105](#) of file [vpbe.h](#).

9.21.2.9 int ipkey

PBE calculation type (this is a cached copy it should not be used directly in code)

Definition at line [122](#) of file [vpbe.h](#).

9.21.2.10 double L

Length of the membrane (A)

Definition at line [132](#) of file [vpbe.h](#).

9.21.2.11 double maxlonRadius

Max ion radius (A; used for calculating accessibility and defining volumes for ionic strength coefficients)

Definition at line [100](#) of file [vpbe.h](#).

9.21.2.12 double membraneDiel

Membrane dielectric constant

Definition at line [133](#) of file [vpbe.h](#).

9.21.2.13 int numlon

Total number of ion species

Definition at line [103](#) of file [vpbe.h](#).

9.21.2.14 int param2Flag

Check to see if bcfl=3 parms have been set

Definition at line [135](#) of file [vpbe.h](#).

9.21.2.15 int paramFlag

Check to see if the parameters have been set

Definition at line [125](#) of file [vpbe.h](#).

9.21.2.16 double smsize

Size-Modified PBE size

Definition at line [121](#) of file [vpbe.h](#).

9.21.2.17 double smvolume

Size-Modified PBE relative volume

Definition at line [120](#) of file [vpbe.h](#).

9.21.2.18 double soluteCenter[3]

Center of solute molecule (A)

Definition at line [113](#) of file [vpbe.h](#).

9.21.2.19 double soluteCharge

Charge of solute molecule (e)

Definition at line [118](#) of file [vpbe.h](#).

9.21.2.20 double soluteDiel

Solute dielectric constant (unitless)

Definition at line [93](#) of file [vpbe.h](#).

9.21.2.21 double soluteRadius

Radius of solute molecule (A)

Definition at line [114](#) of file [vpbe.h](#).

9.21.2.22 double soluteXlen

Solute length in x-direction

Definition at line [115](#) of file [vpbe.h](#).

9.21.2.23 double soluteYlen

Solute length in y-direction

Definition at line [116](#) of file [vpbe.h](#).

9.21.2.24 double soluteZlen

Solute length in z-direction

Definition at line [117](#) of file [vpbe.h](#).

9.21.2.25 double solventDiel

Solvent dielectric constant (unitless)

Definition at line [94](#) of file [vpbe.h](#).

9.21.2.26 double solventRadius

Solvent probe radius (angstroms) for accessibility; determining defining volumes for the dielectric coefficient

Definition at line [96](#) of file [vpbe.h](#).

9.21.2.27 double T

Temperature (K)

Definition at line [92](#) of file [vpbe.h](#).

9.21.2.28 double V

Membrane potential

Definition at line [134](#) of file [vpbe.h](#).

9.21.2.29 Vmem* vmem

Memory management object

Definition at line [86](#) of file [vpbe.h](#).

9.21.2.30 double xkappa

Debye-Huckel parameter (bulk)

Definition at line 108 of file [vpbe.h](#).

9.21.2.31 double z_mem

Z value of the bottom of the membrane (A)

Definition at line 131 of file [vpbe.h](#).

9.21.2.32 double zkappa2

Square of modified Debye-Huckel parameter (bulk)

Definition at line 110 of file [vpbe.h](#).

9.21.2.33 double zmagic

Delta function scaling parameter

Definition at line 111 of file [vpbe.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[vpbe.h](#)

9.22 sVpee Struct Reference

Contains public data members for Vpee class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/fem/apbs/vpee.h>
```

Data Fields

- Gem * [gm](#)
- int [localPartID](#)
- double [localPartCenter](#) [3]
- double [localPartRadius](#)
- int [killFlag](#)

- double [killParam](#)
- Vmem * [mem](#)

9.22.1 Detailed Description

Contains public data members for Vpee class/module.

Author

Nathan Baker

Definition at line [88](#) of file [vpee.h](#).

9.22.2 Field Documentation

9.22.2.1 Gem* gm

Grid manager

Definition at line [90](#) of file [vpee.h](#).

9.22.2.2 int killFlag

A flag indicating the method we're using to artificially decrease the error estimate outside the local partition

Definition at line [98](#) of file [vpee.h](#).

9.22.2.3 double killParam

A parameter for the error estimate attenuation method

Definition at line [101](#) of file [vpee.h](#).

9.22.2.4 double localPartCenter[3]

The coordinates of the center of the local partition

Definition at line [94](#) of file [vpee.h](#).

9.22.2.5 int localPartID

The local partition ID: i.e. the partition whose boundary simplices we're keeping track of

Definition at line 91 of file [vpee.h](#).

9.22.2.6 double localPartRadius

The radius of the circle/sphere which circumscribes the local partition

Definition at line 96 of file [vpee.h](#).

9.22.2.7 Vmem* mem

Memory manager

Definition at line 103 of file [vpee.h](#).

The documentation for this struct was generated from the following file:

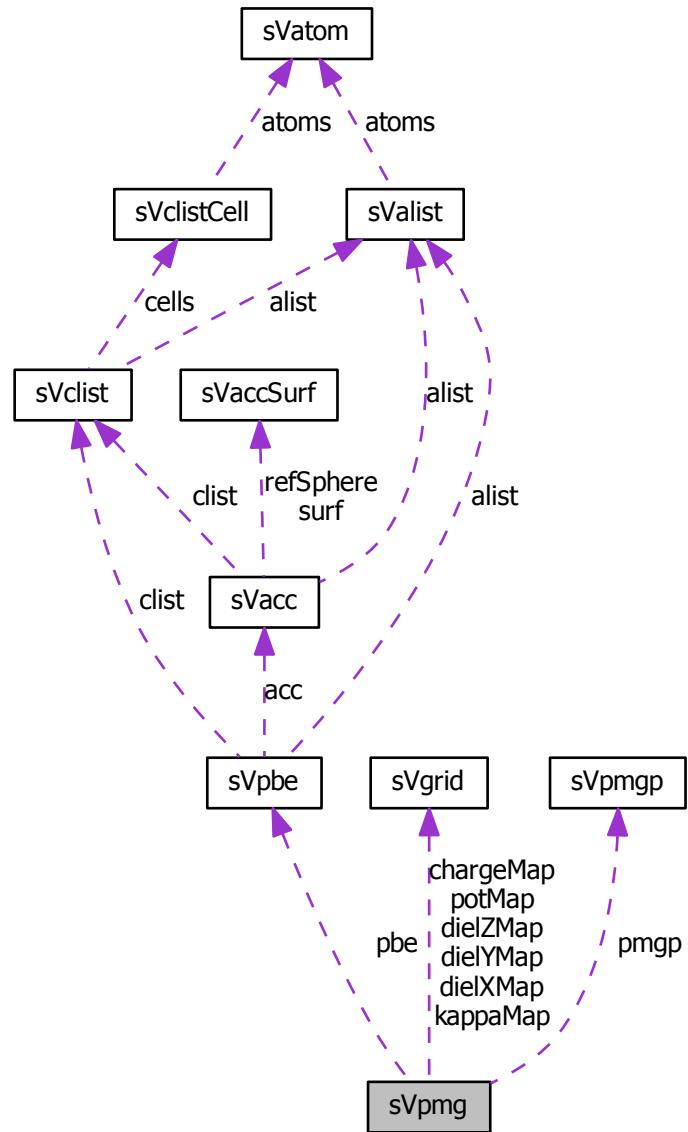
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/[vpee.h](#)

9.23 sVpmg Struct Reference

Contains public data members for Vpmg class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/mg/apbs/vpmg.h>
```

Collaboration diagram for sVpmg:



Data Fields

- Vmem * vmem
- Vpmgp * pmgp
- Vpbe * pbe
- double * epsx
- double * epsy
- double * epsz
- double * kappa
- double * pot
- double * charge
- int * iparm
- double * rparm
- int * iwork
- double * rwork
- double * a1cf
- double * a2cf
- double * a3cf
- double * ccf
- double * fcf
- double * tcf
- double * u
- double * xf
- double * yf
- double * zf
- double * gxcf
- double * gycf
- double * gzcf
- double * pvec
- double extDiEnergy
- double extQmEnergy
- double extQfEnergy
- double extNpEnergy
- Vsurf_Meth surfMeth
- double splineWin
- Vchrg_Meth chargeMeth
- Vchrg_Src chargeSrc
- int filled
- int useDielXMap
- Vgrid * dielXMap
- int useDielYMap
- Vgrid * dielYMap
- int useDielZMap

- `Vgrid * dieIzMap`
- `int useKappaMap`
- `Vgrid * kappaMap`
- `int usePotMap`
- `Vgrid * potMap`
- `int useChargeMap`
- `Vgrid * chargeMap`

9.23.1 Detailed Description

Contains public data members for Vpmg class/module.

Author

Nathan Baker Many of the routines and macros are borrowed from the main.c driver (written by Mike Holst) provided with the PMG code.

Definition at line 96 of file [vpmg.h](#).

9.23.2 Field Documentation

9.23.2.1 double* a1cf

Operator coefficient values (a11) -- this array can be overwritten

Definition at line 113 of file [vpmg.h](#).

9.23.2.2 double* a2cf

Operator coefficient values (a22) -- this array can be overwritten

Definition at line 115 of file [vpmg.h](#).

9.23.2.3 double* a3cf

Operator coefficient values (a33) -- this array can be overwritten

Definition at line 117 of file [vpmg.h](#).

9.23.2.4 double* ccf

Helmholtz term -- this array can be overwritten

Definition at line 119 of file [vpmg.h](#).

9.23.2.5 double* charge

Charge map

Definition at line 107 of file [vpmg.h](#).

9.23.2.6 Vgrid* chargeMap

External charge distribution map

Definition at line 163 of file [vpmg.h](#).

9.23.2.7 Vchrg_Meth chargeMeth

Charge discretization method

Definition at line 140 of file [vpmg.h](#).

9.23.2.8 Vchrg_Src chargeSrc

Charge source

Definition at line 141 of file [vpmg.h](#).

9.23.2.9 Vgrid* dielXMap

External x-shifted dielectric map

Definition at line 147 of file [vpmg.h](#).

9.23.2.10 Vgrid* dielYMap

External y-shifted dielectric map

Definition at line 150 of file [vpmg.h](#).

9.23.2.11 Vgrid* dielZMap

External z-shifted dielectric map

Definition at line 153 of file [vpmg.h](#).

9.23.2.12 double* epsx

X-shifted dielectric map

Definition at line 102 of file [vpmg.h](#).

9.23.2.13 double* epsy

Y-shifted dielectric map

Definition at line 103 of file [vpmg.h](#).

9.23.2.14 double* epsz

Z-shifted dielectric map

Definition at line 104 of file [vpmg.h](#).

9.23.2.15 double extDiEnergy

Stores contributions to the dielectric energy from regions outside the problem domain

Definition at line 130 of file [vpmg.h](#).

9.23.2.16 double extNpEnergy

Stores contributions to the apolar energy from regions outside the problem domain

Definition at line 136 of file [vpmg.h](#).

9.23.2.17 double extQfEnergy

Stores contributions to the fixed charge energy from regions outside the problem domain

Definition at line 134 of file [vpmg.h](#).

9.23.2.18 double extQmEnergy

Stores contributions to the mobile ion energy from regions outside the problem domain

Definition at line 132 of file [vpmg.h](#).

9.23.2.19 double* fcf

Right-hand side -- this array can be overwritten

Definition at line [120](#) of file [vpmg.h](#).

9.23.2.20 int filled

Indicates whether Vpmg_fillco has been called

Definition at line [143](#) of file [vpmg.h](#).

9.23.2.21 double* gxcf

Boundary conditions for x faces

Definition at line [126](#) of file [vpmg.h](#).

9.23.2.22 double* gycf

Boundary conditions for y faces

Definition at line [127](#) of file [vpmg.h](#).

9.23.2.23 double* gzcf

Boundary conditions for z faces

Definition at line [128](#) of file [vpmg.h](#).

9.23.2.24 int* iparm

Passing int parameters to FORTRAN

Definition at line [109](#) of file [vpmg.h](#).

9.23.2.25 int* iwork

Work array

Definition at line [111](#) of file [vpmg.h](#).

9.23.2.26 double* kappa

Ion accessibility map ($0 \leq \text{kappa}(x) \leq 1$)

Definition at line 105 of file [vpmg.h](#).

9.23.2.27 Vgrid* kappaMap

External kappa map

Definition at line 156 of file [vpmg.h](#).

9.23.2.28 Vpbe* pbe

Information about the PBE system

Definition at line 100 of file [vpmg.h](#).

9.23.2.29 Vpmgp* pmgp

Parameters

Definition at line 99 of file [vpmg.h](#).

9.23.2.30 double* pot

Potential map

Definition at line 106 of file [vpmg.h](#).

9.23.2.31 Vgrid* potMap

External potential map

Definition at line 159 of file [vpmg.h](#).

9.23.2.32 double* pvec

Partition mask array

Definition at line 129 of file [vpmg.h](#).

9.23.2.33 double* rparm

Passing real parameters to FORTRAN

Definition at line 110 of file [vpmg.h](#).

9.23.2.34 double* rwork

Work array

Definition at line 112 of file [vpmg.h](#).

9.23.2.35 double splineWin

Spline window parm for surf defs

Definition at line 139 of file [vpmg.h](#).

9.23.2.36 Vsurf_Meth surfMeth

Surface definition method

Definition at line 138 of file [vpmg.h](#).

9.23.2.37 double* tcf

True solution

Definition at line 121 of file [vpmg.h](#).

9.23.2.38 double* u

Solution

Definition at line 122 of file [vpmg.h](#).

9.23.2.39 int useChargeMap

Indicates whether Vpmg_fillco was called with an external charge distribution map

Definition at line 161 of file [vpmg.h](#).

9.23.2.40 int useDielXMap

Indicates whether Vpmg_fillco was called with an external x-shifted dielectric map

Definition at line [145](#) of file [vpmg.h](#).

9.23.2.41 int useDielYMap

Indicates whether Vpmg_fillco was called with an external y-shifted dielectric map

Definition at line [148](#) of file [vpmg.h](#).

9.23.2.42 int useDielZMap

Indicates whether Vpmg_fillco was called with an external z-shifted dielectric map

Definition at line [151](#) of file [vpmg.h](#).

9.23.2.43 int useKappaMap

Indicates whether Vpmg_fillco was called with an external kappa map

Definition at line [154](#) of file [vpmg.h](#).

9.23.2.44 int usePotMap

Indicates whether Vpmg_fillco was called with an external potential map

Definition at line [157](#) of file [vpmg.h](#).

9.23.2.45 Vmem* vmem

Memory management object for this class

Definition at line [98](#) of file [vpmg.h](#).

9.23.2.46 double* xf

Mesh point x coordinates

Definition at line [123](#) of file [vpmg.h](#).

9.23.2.47 double* yf

Mesh point y coordinates

Definition at line 124 of file [vpmgp.h](#).

9.23.2.48 double* zf

Mesh point z coordinates

Definition at line 125 of file [vpmgp.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/[vpmgp.h](#)

9.24 sVpmgp Struct Reference

Contains public data members for Vpmgp class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/mg/apbs/vpmgp.h>
```

Data Fields

- int [nx](#)
- int [ny](#)
- int [nz](#)
- int [nlev](#)
- double [hx](#)
- double [hy](#)
- double [hzed](#)
- int [nonlin](#)
- int [nxc](#)
- int [nyc](#)
- int [nzc](#)
- int [nf](#)
- int [nc](#)
- int [narrc](#)
- int [n_rpc](#)
- int [n_iz](#)
- int [n_ipc](#)

- int `nrwk`
- int `niwk`
- int `narr`
- int `ipkey`
- double `xcent`
- double `ycent`
- double `zcent`
- double `errtol`
- int `itmax`
- int `istop`
- int `iinfo`
- `Vbcfl bcfl`
- int `key`
- int `iperf`
- int `meth`
- int `mgkey`
- int `nu1`
- int `nu2`
- int `mgsmoo`
- int `mgprol`
- int `mgcoar`
- int `mgsolv`
- int `mgdisc`
- double `omegal`
- double `omegan`
- int `irite`
- int `ipcon`
- double `xlen`
- double `ylen`
- double `zlen`
- double `xmin`
- double `ymin`
- double `zmin`
- double `xmax`
- double `ymax`
- double `zmax`

9.24.1 Detailed Description

Contains public data members for Vpmgp class/module.

Author

Nathan Baker

Bug

Value ipcon does not currently allow for preconditioning in PMG

Definition at line [78](#) of file [vpmgp.h](#).

9.24.2 Field Documentation

9.24.2.1 Vbcfl bcfl

Boundary condition method [default = BCFL_SDH]

Definition at line [133](#) of file [vpmgp.h](#).

9.24.2.2 double errtol

Desired error tolerance [default = 1e-9]

Definition at line [119](#) of file [vpmgp.h](#).

9.24.2.3 double hx

Grid x spacings [no default]

Definition at line [85](#) of file [vpmgp.h](#).

9.24.2.4 double hy

Grid y spacings [no default]

Definition at line [86](#) of file [vpmgp.h](#).

9.24.2.5 double hzed

Grid z spacings [no default]

Definition at line [87](#) of file [vpmgp.h](#).

9.24.2.6 int iinfo

Runtime status messages [default = 1]

- 0: none
- 1: some
- 2: lots
- 3: more

Definition at line 128 of file [vpmgp.h](#).

9.24.2.7 int ipcon

Preconditioning method [default = 3]

- 0: diagonal
- 1: ICCG
- 2: ICCGDW
- 3: MICCGDW
- 4: none

Definition at line 181 of file [vpmgp.h](#).

9.24.2.8 int iperf

Analysis of the operator [default = 0]

- 0: no
- 1: condition number
- 2: spectral radius
- 3: cond. number & spectral radius

Definition at line 137 of file [vpmgp.h](#).

9.24.2.9 int ipkey

Toggles nonlinearity (set by nonlin)

- -2: Size-Modified PBE
- -1: Linearized PBE
- 0: Nonlinear PBE with capped sinh term [default]
- >1: Polynomial approximation to sinh, note that ipkey must be odd

Definition at line 107 of file [vpmgp.h](#).

9.24.2.10 int irite

FORTRAN output unit [default = 8]

Definition at line 180 of file [vpmgp.h](#).

9.24.2.11 int istop

Stopping criterion [default = 1]

- 0: residual
- 1: relative residual
- 2: diff
- 3: errc
- 4: errd
- 5: aerrd

Definition at line 121 of file [vpmgp.h](#).

9.24.2.12 int itmax

Maximum number of iters [default = 100]

Definition at line 120 of file [vpmgp.h](#).

9.24.2.13 int key

Print solution to file [default = 0]

- 0: no
- 1: yes

Definition at line 134 of file [vpmgp.h](#).

9.24.2.14 int meth

Solution method [default = 2]

- 0: conjugate gradient multigrid
- 1: newton
- 2: multigrid
- 3: conjugate gradient
- 4: successive overrelaxation
- 5: red-black gauss-seidel
- 6: weighted jacobi
- 7: richardson
- 8: conjugate gradient multigrid aqua
- 9: newton aqua

Definition at line 142 of file [vpmgp.h](#).

9.24.2.15 int mgcoar

Coarsening method [default = 2]

- 0: standard
- 1: harmonic
- 2: galerkin

Definition at line 168 of file [vpmgp.h](#).

9.24.2.16 int mgdisc

Discretization method [default = 0]

- 0: finite volume
- 1: finite element

Definition at line 175 of file [vpmgp.h](#).

9.24.2.17 int mgkey

Multigrid method [default = 0]

- 0: variable v-cycle
- 1: nested iteration

Definition at line 153 of file [vpmgp.h](#).

9.24.2.18 int mgprol

Prolongation method [default = 0]

- 0: trilinear
- 1: operator-based
- 2: mod. operator-based

Definition at line 164 of file [vpmgp.h](#).

9.24.2.19 int mgsmoo

Smoothing method [default = 1]

- 0: weighted jacobi
- 1: gauss-seidel
- 2: SOR
- 3: richardson
- 4: cgls

Definition at line 158 of file [vpmgp.h](#).

9.24.2.20 int mgsolv

Coarse equation solve method [default = 1]

- 0: cgns
- 1: banded linpack

Definition at line 172 of file [vpmgp.h](#).

9.24.2.21 int n_ipc

Integer info work array required storage

Definition at line 102 of file [vpmgp.h](#).

9.24.2.22 int n_iz

Integer storage parameter (index max)

Definition at line 101 of file [vpmgp.h](#).

9.24.2.23 int n_rpc

Real info work array required storage

Definition at line 100 of file [vpmgp.h](#).

9.24.2.24 int narr

Array work storage

Definition at line 106 of file [vpmgp.h](#).

9.24.2.25 int narrc

Size of vector on coarse level

Definition at line 99 of file [vpmgp.h](#).

9.24.2.26 int nc

Number of coarse grid unknowns

Definition at line 98 of file [vpmgp.h](#).

9.24.2.27 int nf

Number of fine grid unknowns

Definition at line 97 of file [vpmgp.h](#).

9.24.2.28 int niwk

Integer work storage

Definition at line 105 of file [vpmgp.h](#).

9.24.2.29 int nlev

Number of mesh levels [no default]

Definition at line 84 of file [vpmgp.h](#).

9.24.2.30 int nonlin

Problem type [no default]

- 0: linear
- 1: nonlinear
- 2: linear then nonlinear

Definition at line 88 of file [vpmgp.h](#).

9.24.2.31 int nrwk

Real work storage

Definition at line 104 of file [vpmgp.h](#).

9.24.2.32 int nu1

Number of pre-smoothings [default = 2]

Definition at line 156 of file [vpmgp.h](#).

9.24.2.33 int nu2

Number of post-smoothings [default = 2]

Definition at line 157 of file [vpmgp.h](#).

9.24.2.34 int nx

Grid x dimensions [no default]

Definition at line 81 of file [vpmgp.h](#).

9.24.2.35 int nxc

Coarse level grid x dimensions

Definition at line 94 of file [vpmgp.h](#).

9.24.2.36 int ny

Grid y dimensions [no default]

Definition at line 82 of file [vpmgp.h](#).

9.24.2.37 int nyc

Coarse level grid y dimensions

Definition at line 95 of file [vpmgp.h](#).

9.24.2.38 int nz

Grid z dimensions [no default]

Definition at line 83 of file [vpmgp.h](#).

9.24.2.39 int nzc

Coarse level grid z dimensions

Definition at line 96 of file [vpmgp.h](#).

9.24.2.40 double omegal

Linear relax parameter [default = 8e-1]

Definition at line 178 of file [vpmgp.h](#).

9.24.2.41 double omegan

Nonlin relax parameter [default = 9e-1]

Definition at line 179 of file [vpmgp.h](#).

9.24.2.42 double xcent

Grid x center [0]

Definition at line 116 of file [vpmgp.h](#).

9.24.2.43 double xlabel

Domain x length

Definition at line 187 of file [vpmgp.h](#).

9.24.2.44 double xmax

Domain upper x corner

Definition at line 193 of file [vpmgp.h](#).

9.24.2.45 double xmin

Domain lower x corner

Definition at line 190 of file [vpmgp.h](#).

9.24.2.46 double ycent

Grid y center [0]

Definition at line 117 of file [vpmgp.h](#).

9.24.2.47 double ylen

Domain y length

Definition at line 188 of file [vpmgp.h](#).

9.24.2.48 double ymax

Domain upper y corner

Definition at line 194 of file [vpmgp.h](#).

9.24.2.49 double ymin

Domain lower y corner

Definition at line 191 of file [vpmgp.h](#).

9.24.2.50 double zcent

Grid z center [0]

Definition at line 118 of file [vpmgp.h](#).

9.24.2.51 double zlen

Domain z length

Definition at line 189 of file [vpmgp.h](#).

9.24.2.52 double zmax

Domain upper z corner

Definition at line 195 of file [vpmgp.h](#).

9.24.2.53 double zmin

Domain lower z corner

Definition at line 192 of file [vpmgp.h](#).

The documentation for this struct was generated from the following file:

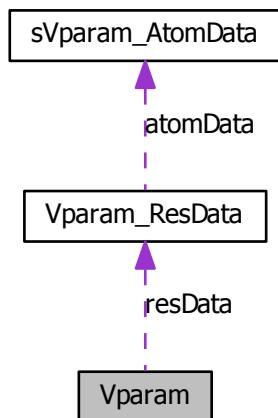
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/[vpmgp.h](#)

9.25 Vparam Struct Reference

Reads and assigns charge/radii parameters.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source  
code/APBS/src/generic/apbs/vparam.h>
```

Collaboration diagram for Vparam:



Data Fields

- Vmem * vmem
- int nResData
- [Vparam_ResData](#) * resData

9.25.1 Detailed Description

Reads and assigns charge/radii parameters.

Author

Nathan Baker

Definition at line 130 of file [vparam.h](#).

9.25.2 Field Documentation

9.25.2.1 int nResData

Number of [Vparam_ResData](#) objects associated with this object

Definition at line 133 of file [vparam.h](#).

9.25.2.2 Vparam_ResData* resData

Array of nResData [Vparam_ResData](#) objects

Definition at line 135 of file [vparam.h](#).

9.25.2.3 Vmem* vmem

Memory management object for this class

Definition at line 132 of file [vparam.h](#).

The documentation for this struct was generated from the following file:

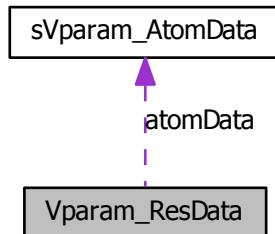
- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[vparam.h](#)

9.26 Vparam_ResData Struct Reference

ResData sub-class; stores residue data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vparam.h>
```

Collaboration diagram for Vparam_ResData:



Data Fields

- Vmem * [vmem](#)
- char [name](#) [VMAX_ARGLEN]
- int [nAtomData](#)
- [Vparam_AtomData](#) * [atomData](#)

9.26.1 Detailed Description

ResData sub-class; stores residue data.

Author

Nathan Baker

Definition at line [109](#) of file [vpParam.h](#).

9.26.2 Field Documentation

9.26.2.1 [Vparam_AtomData](#)* [atomData](#)

Array of Vparam_AtomData natom objects

Definition at line [114](#) of file [vpParam.h](#).

9.26.2.2 char name[VMAX_ARGLEN]

Residue name

Definition at line 111 of file [vparam.h](#).

9.26.2.3 int nAtomData

Number of Vparam_AtomData objects associated with this object

Definition at line 112 of file [vparam.h](#).

9.26.2.4 Vmem* vmem

Pointer to memory manager from [Vparam](#) master class

Definition at line 110 of file [vparam.h](#).

The documentation for this struct was generated from the following file:

- C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/[vparam.h](#)

Chapter 10

File Documentation

10.1 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/doc/license/LICENSE.h File Reference

APBS license.

10.1.1 Detailed Description

APBS license.

Author

Nathan Baker

Version

Id:

LICENSE.h 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*
```

```
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
```

Definition in file [LICENSE.h](#).

10.2 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/doc/license/LICENSE.h

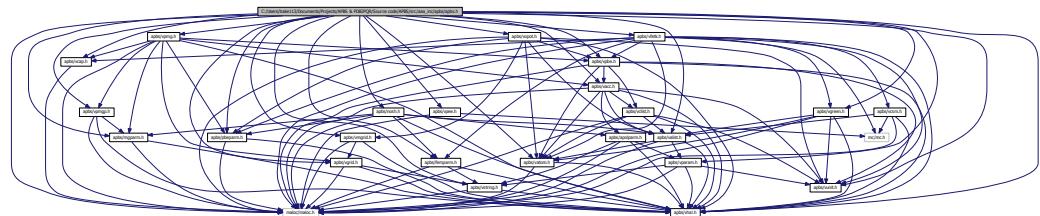
00001

10.3 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/aaa_inc/apbs/apbs.h File Reference

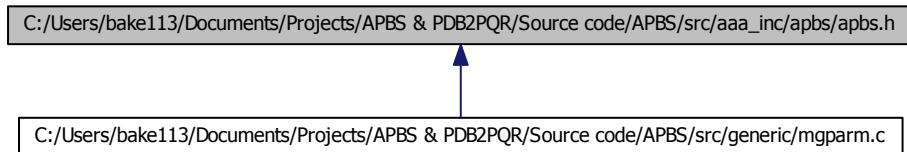
Top-level header for APBS.

```
#include "maloc/maloc.h"
#include "apbs/femparm.h"
#include "apbs/mgparm.h"
#include "apbs/nosh.h"
#include "apbs/pbeparm.h"
#include "apbs/vacc.h"
#include "apbs/valist.h"
#include "apbs/vatom.h"
#include "apbs/vcap.h"
#include "apbs/vgreen.h"
#include "apbs/vhal.h"
#include "apbs/vpbe.h"
#include "apbs/vstring.h"
#include "apbs/vunit.h"
#include "apbs/vparam.h"
#include "apbs/vgrid.h"
#include "apbs/vmgrid.h"
#include "apbs/vopot.h"
#include "apbs/vpmg.h"
#include "apbs/vpmgp.h"
#include "apbs/vfetk.h"
#include "apbs/vpee.h"
```

Include dependency graph for apbs.h:



This graph shows which files directly or indirectly include this file:



10.3.1 Detailed Description

Top-level header for APBS.

Version

Id:

[apbs.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*
```

```
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
```

Definition in file [apbs.h](#).

10.4 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/aaa_inc/apbs/apbs.h

```
00001
00057 #ifndef _APBS_H_
00058 #define _APBS_H_
00059
```

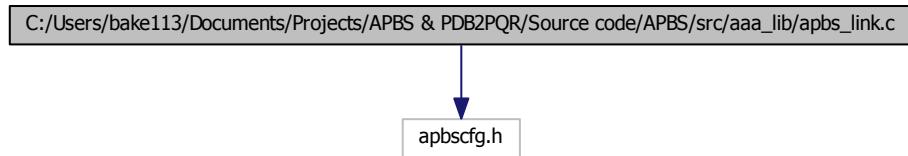
```
00060 /* MALOC headers */
00061 #include "maloc/maloc.h"
00062
00063 /* Generic headers */
00064 #include "apbs/femparm.h"
00065 #include "apbs/mgparm.h"
00066 #include "apbs/nosh.h"
00067 #include "apbs/pbeparm.h"
00068 #include "apbs/vacc.h"
00069 #include "apbs/valist.h"
00070 #include "apbs/vatom.h"
00071 #include "apbs/vcap.h"
00072 #include "apbs/vgreen.h"
00073 #include "apbs/vhal.h"
00074 #include "apbs/vpbe.h"
00075 #include "apbs/vstring.h"
00076 #include "apbs/vunit.h"
00077 #include "apbs/vparam.h"
00078
00079 /* MG headers */
00080 #include "apbs/vgrid.h"
00081 #include "apbs/vmgrid.h"
00082 #include "apbs/vopot.h"
00083 #include "apbs/vpmg.h"
00084 #include "apbs/vpmgp.h"
00085
00086 /* FEM headers */
00087 #if defined(HAVE_MC_H)
00088 # include "apbs/vfetk.h"
00089 # include "apbs/vpee.h"
00090 #endif
00091
00092 #endif /* ifndef _APBS_H_ */
```

10.5 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/aaa_lib/apbs_link.c File Reference

Autoconf linkage assistance for packages built on top of APBS.

```
#include "apbscfg.h"
```

Include dependency graph for apbs_link.c:



Functions

- void **apbs_needs_mc** (void)
- void **apbs_needs_blas** (void)
- void **apbs_link** (void)

10.5.1 Detailed Description

Autoconf linkage assistance for packages built on top of APBS.

Author

Nathan Baker and Michael Holst

Version

Id:

[apbs_link.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial
```

```

* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [apbs_link.c](#).

10.6 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/aaa_lib/apbs_link.c

```

00001 #include "apbscfg.h"
00002
00058 #if defined(HAVE_MC_H)
00059     void apbs_needs_mc(void) { }
00060 #endif
00061 #if !defined(USE_PMG_BLAS)
00062     void apbs_needs_blas(void) { }
00063 #endif
00064
00065 void apbs_link(void)
00066 {

```

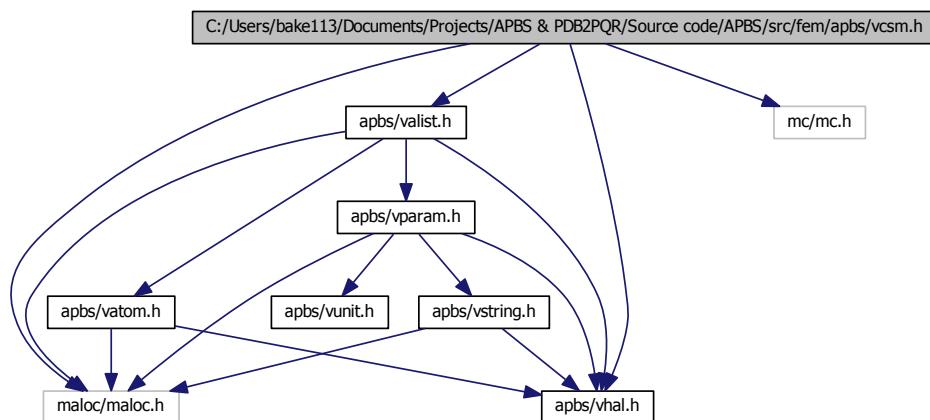
```
00067 }  
00068
```

10.7 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/vcsm.h File Reference

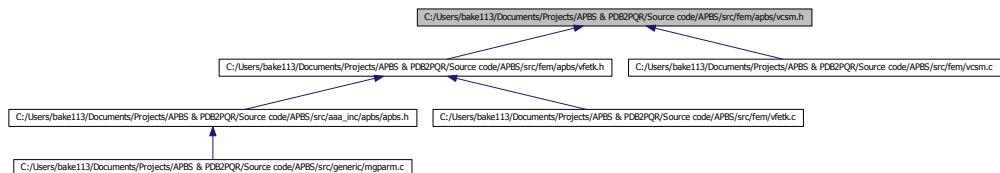
Contains declarations for the Vcsm class.

```
#include "maloc/malloc.h"  
#include "apbs/vhal.h"  
#include "apbs/valist.h"  
#include "mc/mc.h"
```

Include dependency graph for vcsm.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **sVcsm**
Charge-simplex map class.

Typedefs

- typedef struct **sVcsm Vcsm**
Declaration of the Vcsm class as the Vcsm structure.

Functions

- VEXTERNC void **Gem_setExternalUpdateFunction** (Gem *thee, void(*externalUpdate)(SS **simp, int num))
External function for FEtk Gem class to use during mesh refinement.
- VEXTERNC **Valist * Vcsm_getValist** (**Vcsm** *thee)
Get atom list.
- VEXTERNC int **Vcsm_getNumberAtoms** (**Vcsm** *thee, int isimp)
Get number of atoms associated with a simplex.
- VEXTERNC **Vatom * Vcsm_getAtom** (**Vcsm** *thee, int iatom, int isimp)
Get particular atom associated with a simplex.
- VEXTERNC int **Vcsm_getAtomIndex** (**Vcsm** *thee, int iatom, int isimp)
Get ID of particular atom in a simplex.
- VEXTERNC int **Vcsm_getNumberSimplices** (**Vcsm** *thee, int iatom)
Get number of simplices associated with an atom.
- VEXTERNC SS * **Vcsm_getSimplex** (**Vcsm** *thee, int isimp, int iatom)
Get particular simplex associated with an atom.
- VEXTERNC int **Vcsm_getSimplexIndex** (**Vcsm** *thee, int isimp, int iatom)
Get index particular simplex associated with an atom.

- VEXTERNC unsigned long int [Vcsm_memChk](#) ([Vcsm](#) *thee)
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC [Vcsm](#) * [Vcsm_ctor](#) ([Valist](#) *alist, [Gem](#) *gm)
Construct Vcsm object.
- VEXTERNC int [Vcsm_ctor2](#) ([Vcsm](#) *thee, [Valist](#) *alist, [Gem](#) *gm)
FORTRAN stub to construct Vcsm object.
- VEXTERNC void [Vcsm_dtor](#) ([Vcsm](#) **thee)
Destroy Vcsm object.
- VEXTERNC void [Vcsm_dtor2](#) ([Vcsm](#) *thee)
FORTRAN stub to destroy Vcsm object.
- VEXTERNC void [Vcsm_init](#) ([Vcsm](#) *thee)
Initialize charge-simplex map with mesh and atom data.
- VEXTERNC int [Vcsm_update](#) ([Vcsm](#) *thee, [SS](#) **simps, int num)
Update the charge-simplex and simplex-charge maps after refinement.

10.7.1 Detailed Description

Contains declarations for the [Vcsm](#) class.

Version

Id:

[vcsm.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of
```

```

* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vcsm.h](#).

10.8 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/vcsm.h

```

00001
00063 #ifndef _VCSTM_H_
00064 #define _VCSTM_H_
00065
00066 /* Generic headers */
00067 #include "maloc/maloc.h"
00068 #include "apbs/vhal.h"
00069 #include "apbs/valist.h"
00070
00071 /* Specific headers */
00072 #include "mc/mc.h"
00073
00078 VEXTERNC void Gem_setExternalUpdateFunction(
00079     Gem *thee,
00080     void (*externalUpdate)(SS **simps, int num)
00083 );

```

```
00084
00089 struct sVcsm {
00090
00091     Valist *alist;
00092     int natom;
00094     Gem *gm;
00097     int **sqm;
00104     int *nsqm;
00105     int nsimp;
00107     int msimp;
00109     int **qsm;
00111     int *nqsm;
00112     int initFlag;
00114     Vmem *vmem;
00116 };
00117
00122 typedef struct sVcsm Vcsm;
00123
00124 /* //////////////////////////////// Class Vcsm: Inlineable methods (vcsm.c)
00125 // Class Vcsm: Inlineable methods (vcsm.c)
00127
00128 #if !defined(VINLINE_VCSM)
00129
00135     VEXTERNC Valist* Vcsm_getValist(
00136         Vcsm *thee
00137     );
00138
00144     VEXTERNC int Vcsm_getNumberAtoms(
00145         Vcsm *thee,
00146         int isimp
00147     );
00148
00154     VEXTERNC Vatom* Vcsm_getAtom(
00155         Vcsm *thee,
00156         int iatom,
00157         int isimp
00158     );
00159
00165     VEXTERNC int Vcsm_getAtomIndex(
00166         Vcsm *thee,
00167         int iatom,
00168         int isimp
00169     );
00170
00176     VEXTERNC int Vcsm_getNumberSimplices(
00177         Vcsm *thee,
00178         int iatom
00179     );
00180
00186     VEXTERNC SS* Vcsm_getSimplex(
00187         Vcsm *thee,
00188         int isimp,
00189         int iatom
00190     );
00191
00197     VEXTERNC int Vcsm_getSimplexIndex(
00198         Vcsm *thee,
```

```

00199         int isimp,
00200         int iatom
00201     );
00202
00203 VEXTERNC unsigned long int Vcsm_memChk(
00204     Vcsm *thee
00205 );
00206
00207 #else /* if defined(VINLINE_VCSM) */
00208 # define Vcsm_getValist(thee) ((thee)->alist)
00209 # define Vcsm_getNumberAtoms(thee, isimp) ((thee)->nsgm[isimp])
00210 # define Vcsm_getAtom(thee, iatom, isimp) (Valist_getAtom((thee)->alist, ((thee)
00211     ->sgm)[isimp][iatom]))
00212 # define Vcsm_getAtomIndex(thee, iatom, isimp) (((thee)->sgm)[isimp][iatom])
00213 # define Vcsm_getNumberSimplices(thee, iatom) (((thee)->nqsm)[iatom])
00214 # define Vcsm_getSimplex(thee, isimp, iatom) (Gem_SS((thee)->gm, ((thee)->qsm)[
00215     iatom][isimp]))
00216 # define Vcsm_getSimplexIndex(thee, isimp, iatom) (((thee)->qsm)[iatom][isimp])
00217
00218 # define Vcsm_memChk(thee) (Vmem_bytes((thee)->vmem))
00219 #endif /* if !defined(VINLINE_VCSM) */
00220
00221 /* //////////////////////////////// */
00222 // Class Vcsm: Non-Inlineable methods (vcsms.c)
00223
00224
00225 VEXTERNC Vcsm* Vcsm_ctor(
00226     Valist *alist,
00227     Gem *gm
00228 );
00229
00230
00231 VEXTERNC int Vcsm_ctor2(
00232     Vcsm *thee,
00233     Valist *alist,
00234     Gem *gm
00235 );
00236
00237
00238 VEXTERNC void Vcsm_dtor(
00239     Vcsm **thee
00240 );
00241
00242
00243 VEXTERNC void Vcsm_dtor2(
00244     Vcsm *thee
00245 );
00246
00247
00248 VEXTERNC void Vcsm_init(
00249     Vcsm *thee
00250 );
00251
00252
00253 VEXTERNC int Vcsm_update(
00254     Vcsm *thee,
00255     SS **simpsons,
00256     int num
00257 );
00258
00259 #endif /* ifndef _VCSM_H_ */

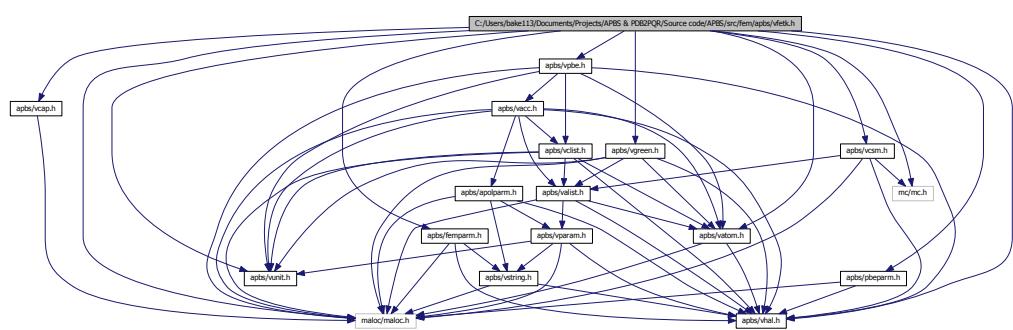
```

10.9 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/vfetk.h File Reference

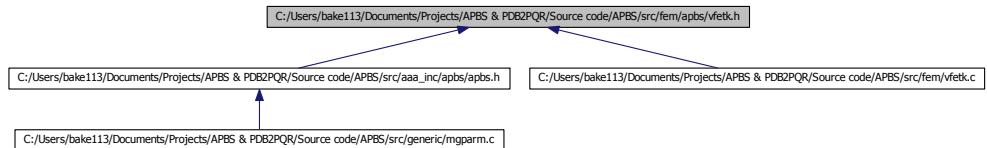
Contains declarations for class Vfetk.

```
#include "maloc/maloc.h"  
#include "mc/mc.h"  
#include "apbs/vhal.h"  
#include "apbs/vatom.h"  
#include "apbs/vcsm.h"  
#include "apbs/vpbe.h"  
#include "apbs/vunit.h"  
#include "apbs/vgreen.h"  
#include "apbs/vcap.h"  
#include "apbs/pbeparm.h"
```

Include dependency graph for vfetk.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **sVfetk**
Contains public data members for Vfetk class/module.
- struct **sVfetk_LocalVar**
Vfetk LocalVar subclass.

Typedefs

- typedef enum **eVfetk_LsolvType** **Vfetk_LsolvType**
Declare FEMparm_LsolvType type.
- typedef enum **eVfetk_MeshLoad** **Vfetk_MeshLoad**
Declare FEMparm_GuessType type.
- typedef enum **eVfetk_NsolvType** **Vfetk_NsolvType**
Declare FEMparm_NsolvType type.
- typedef enum **eVfetk_GuessType** **Vfetk_GuessType**
Declare FEMparm_GuessType type.
- typedef enum **eVfetk_PrecType** **Vfetk_PrecType**
Declare FEMparm_GuessType type.
- typedef struct **sVfetk** **Vfetk**
Declaration of the Vfetk class as the Vfetk structure.
- typedef struct **sVfetk_LocalVar** **Vfetk_LocalVar**
Declaration of the Vfetk_LocalVar subclass as the Vfetk_LocalVar structure.

Enumerations

- enum **eVfetk_LsolvType** { **VLT_SLU** = 0, **VLT_MG** = 1, **VLT(CG** = 2, **VLT_BCG** = 3 }
Linear solver type.

- enum `eVfetk_MeshLoad` { `VML_DIRICUBE`, `VML_NEUMCUBE`, `VML_EXTERNAL` }
- Mesh loading operation.*
- enum `eVfetk_NsolvType` { `VNT_NEW` = 0, `VNT_INC` = 1, `VNT_ARC` = 2 }
- Non-linear solver type.*
- enum `eVfetk_GuessType` { `VGT_ZERO` = 0, `VGT_DIRI` = 1, `VGT_PREV` = 2 }
- Initial guess type.*
- enum `eVfetk_PrecType` { `VPT_IDEN` = 0, `VPT_DIAG` = 1, `VPT_MG` = 2 }
- Preconditioner type.*

Functions

- VEXTERNC `Gem * Vfetk_getGem` (`Vfetk *thee`)
Get a pointer to the Gem (grid manager) object.
- VEXTERNC `AM * Vfetk_getAM` (`Vfetk *thee`)
Get a pointer to the AM (algebra manager) object.
- VEXTERNC `Vpbe * Vfetk_getVpbe` (`Vfetk *thee`)
Get a pointer to the Vpbe (PBE manager) object.
- VEXTERNC `Vcsm * Vfetk_getVcsm` (`Vfetk *thee`)
Get a pointer to the Vcsm (charge-simplex map) object.
- VEXTERNC int `Vfetk_getAtomColor` (`Vfetk *thee`, int `iatom`)
Get the partition information for a particular atom.
- VEXTERNC `Vfetk * Vfetk_ctor` (`Vpbe *pbe`, `Vhal_PBEType` `type`)
Constructor for Vfetk object.
- VEXTERNC int `Vfetk_ctor2` (`Vfetk *thee`, `Vpbe *pbe`, `Vhal_PBEType` `type`)
FORTRAN stub constructor for Vfetk object.
- VEXTERNC void `Vfetk_dtor` (`Vfetk **thee`)
Object destructor.
- VEXTERNC void `Vfetk_dtor2` (`Vfetk *thee`)
FORTRAN stub object destructor.
- VEXTERNC double * `Vfetk_getSolution` (`Vfetk *thee`, int *`length`)
Create an array containing the solution (electrostatic potential in units of $k_B T / e$) at the finest mesh level.
- VEXTERNC void `Vfetk_setParameters` (`Vfetk *thee`, `PBparm *pbparm`, `FEM-parm *feparm`)
Set the parameter objects.
- VEXTERNC double `Vfetk_energy` (`Vfetk *thee`, int `color`, int `nonlin`)
Return the total electrostatic energy.
- VEXTERNC double `Vfetk_dqmEnergy` (`Vfetk *thee`, int `color`)
Get the "mobile charge" and "polarization" contributions to the electrostatic energy.

- VEXTERNC double [Vfetk_qfEnergy](#) ([Vfetk](#) *thee, int color)

Get the "fixed charge" contribution to the electrostatic energy.
- VEXTERNC unsigned long int [Vfetk_memChk](#) ([Vfetk](#) *thee)

Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC void [Vfetk_setAtomColors](#) ([Vfetk](#) *thee)

Transfer color (partition ID) information from a partitioned mesh to the atoms.
- VEXTERNC void [Bmat_printHB](#) (Bmat *thee, char *fname)

Writes a Bmat to disk in Harwell-Boeing sparse matrix format.
- VEXTERNC Vrc_Codes [Vfetk_genCube](#) ([Vfetk](#) *thee, double center[3], double length[3], [Vfetk_MeshLoad](#) meshType)

Construct a rectangular mesh (in the current Vfetk object)
- VEXTERNC Vrc_Codes [Vfetk_loadMesh](#) ([Vfetk](#) *thee, double center[3], double length[3], [Vfetk_MeshLoad](#) meshType, Vio *sock)

Loads a mesh into the Vfetk (and associated) object(s).
- VEXTERNC PDE * [Vfetk_PDE_ctor](#) ([Vfetk](#) *fetk)

Constructs the FEtk PDE object.
- VEXTERNC int [Vfetk_PDE_ctor2](#) (PDE *thee, [Vfetk](#) *fetk)

Initializes the FEtk PDE object.
- VEXTERNC void [Vfetk_PDE_dtor](#) (PDE **thee)

Destroys FEtk PDE object.
- VEXTERNC void [Vfetk_PDE_dtor2](#) (PDE *thee)

FORTRAN stub: destroys FEtk PDE object.
- VEXTERNC void [Vfetk_PDE_initAssemble](#) (PDE *thee, int ip[], double rp[])

Do once-per-assembly initialization.
- VEXTERNC void [Vfetk_PDE_initElement](#) (PDE *thee, int elementType, int chart, double tvx[][VAPBS_DIM], void *data)

Do once-per-element initialization.
- VEXTERNC void [Vfetk_PDE_initFace](#) (PDE *thee, int faceType, int chart, double tvec[])

Do once-per-face initialization.
- VEXTERNC void [Vfetk_PDE_initPoint](#) (PDE *thee, int pointType, int chart, double txq[], double tU[], double tDU[][VAPBS_DIM])

Do once-per-point initialization.
- VEXTERNC void [Vfetk_PDE_Fu](#) (PDE *thee, int key, double F[])

Evaluate strong form of PBE. For interior points, this is:

$$-\nabla \cdot \epsilon \nabla u + b(u) - f$$

where $b(u)$ is the (possibly nonlinear) mobile ion term and f is the source charge distribution term (for PBE) or the induced surface charge distribution (for RPBE). For an interior-boundary (simplex face) point, this is:

$$[\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^+} - [\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^-}$$

where $n(x)$ is the normal to the simplex face and the term represents the jump in dielectric displacement across the face. There is no outer-boundary contribution for this problem.

- VEXTERNC double [Vfetk_PDE_Fu_v](#) (PDE *thee, int key, double V[], double dV[])[VAPBS_DIM])

This is the weak form of the PBE; i.e. the strong form integrated with a test function to give:

$$\int_{\Omega} [\varepsilon \nabla u \cdot \nabla v + b(u)v - fv] dx$$

where $b(u)$ denotes the mobile ion term.

- VEXTERNC double [Vfetk_PDE_DFu_wv](#) (PDE *thee, int key, double W[], double dW[])[VAPBS_DIM], double V[], double dV[])[VAPBS_DIM])

This is the linearization of the weak form of the PBE; e.g., for use in a Newton iteration.

This is the functional linearization of the strong form integrated with a test function to give:

$$\int_{\Omega} [\varepsilon \nabla w \cdot \nabla v + b'(u)wv - fv] dx$$

where $b'(u)$ denotes the functional derivation of the mobile ion term.

- VEXTERNC void [Vfetk_PDE_delta](#) (PDE *thee, int type, int chart, double txq[], void *user, double F[]))

Evaluate a (discretized) delta function source term at the given point.

- VEXTERNC void [Vfetk_PDE_u_D](#) (PDE *thee, int type, int chart, double txq[], double F[]))

Evaluate the Dirichlet boundary condition at the given point.

- VEXTERNC void [Vfetk_PDE_u_T](#) (PDE *thee, int type, int chart, double txq[], double F[]))

Evaluate the "true solution" at the given point for comparison with the numerical solution.

- VEXTERNC void [Vfetk_PDE_bisectEdge](#) (int dim, int dimII, int edgeType, int chart[], double vx[])[VAPBS_DIM])

Define the way manifold edges are bisected.

- VEXTERNC void [Vfetk_PDE_mapBoundary](#) (int dim, int dimII, int vertexType, int chart, double vx[VAPBS_DIM])

Map a boundary point to some pre-defined shape.

- VEXTERNC int [Vfetk_PDE_markSimplex](#) (int dim, int dimII, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[])[VAPBS_DIM], void *simplex)

User-defined error estimator -- in our case, a geometry-based refinement method; forcing simplex refinement at the dielectric boundary and (for non-regularized PBE) the charges.

- VEXTERNC void [Vfetk_PDE_oneChart](#) (int dim, int dimII, int objType, int chart[], double vx[])[VAPBS_DIM], int dimV)

Unify the chart for different coordinate systems -- a no-op for us.

- VEXTERNC double [Vfetk_PDE_Ju](#) (PDE *thee, int key)

Energy functional. This returns the energy (less delta function terms) in the form:

$$c^{-1}/2 \int (\varepsilon(\nabla u)^2 + \kappa^2(\cosh u - 1))dx$$

for a 1:1 electrolyte where c is the output from Vpbe_getZmagic.

- VEXTERNC void [Vfetk_externalUpdateFunction](#) (SS **simps, int num)

External hook to simplex subdivision routines in Gem. Called each time a simplex is subdivided (we use it to update the charge-simplex map)
- VEXTERNC int [Vfetk_PDE_simplexBasisInit](#) (int key, int dim, int comp, int *ndof, int dof[])

Initialize the bases for the trial or the test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VEXTERNC void [Vfetk_PDE_simplexBasisForm](#) (int key, int dim, int comp, int pdkey, double xq[], double basis[])

Evaluate the bases for the trial or test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VEXTERNC void [Vfetk_readMesh](#) (Vfetk *thee, int skey, Vio *sock)

Read in mesh and initialize associated internal structures.
- VEXTERNC void [Vfetk_dumpLocalVar](#) ()

Debugging routine to print out local variables used by PDE object.
- VEXTERNC int [Vfetk_fillArray](#) (Vfetk *thee, Bvec *vec, [Vdata_Type](#) type)

Fill an array with the specified data.
- VEXTERNC int [Vfetk_write](#) (Vfetk *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, Bvec *vec, [Vdata_Format](#) format)

Write out data.
- VEXTERNC Vrc_Codes [Vfetk_loadGem](#) (Vfetk *thee, Gem *gm)

Load a Gem geometry manager object into Vfetk.

10.9.1 Detailed Description

Contains declarations for class Vfetk.

Version

Id:

[vfetk.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [vfetk.h](#).

10.10 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/vfetk.h

```

00001
00062 #ifndef _VFETK_H_
00063 #define _VFETK_H_
00064
00065 #include "malloc/malloc.h"
00066 #include "mc/mc.h"
00067 #include "apbs/vhal.h"
00068 #include "apbs/vatom.h"
00069 /* #include "apbs/valist.h" */
00070 #include "apbs/vcsm.h"
00071 #include "apbs/vpbe.h"
00072 #include "apbs/vunit.h"
00073 #include "apbs/vgreen.h"
00074 #include "apbs/vcap.h"
00075 #include "apbs/pbeparm.h"
00076 #include "apbs/femparm.h"
00077
00083 enum eVfetk_LsolvType {
00084     VLT_SLU=0,
00085     VLT_MG=1,
00086     VLT(CG)=2,
00087     VLT_BCG=3
00088 };
00089
00094 typedef enum eVfetk_LsolvType Vfetk_LsolvType;
00095
00096
00101 enum eVfetk_MeshLoad {
00102     VML_DIRICUBE,
00103     VML_NEUMCUBE,
00104     VML_EXTERNAL
00105 };
00106
00111 typedef enum eVfetk_MeshLoad Vfetk_MeshLoad;
00112
00118 enum eVfetk_NsolvType {
00119     VNT_NEW=0,
00120     VNT_INC=1,
00121     VNT_ARC=2
00122 };
00123
00128 typedef enum eVfetk_NsolvType Vfetk_NsolvType;
00129
00135 enum eVfetk_GuessType {
00136     VGT_ZERO=0,
00137     VGT_DIRI=1,
00138     VGT_PREV=2
00139 };
00140
00145 typedef enum eVfetk_GuessType Vfetk_GuessType;
00146
00152 enum eVfetk_PrecType {
00153     VPT_IDEN=0,

```

```
00154     VPT_DIAG=1,  
00155     VPT_MG=2  
00156 };  
00157  
00162 typedef enum eVfetk_PrecType Vfetk_PrecType;  
00163  
00173 struct svfetk {  
00174  
00175     Vmem *vmem;  
00176     Gem *gm;  
00179     AM *am;  
00180     Aprx *aprx;  
00181     PDE *pde;  
00182     Vpbe *pbe;  
00183     Vcsm *csm;  
00184     Vfetk_LsolvType lkey;  
00185     int lmax;  
00186     double ltol;  
00187     Vfetk_NsolvType nkey;  
00188     int nmax;  
00189     double ntol;  
00190     Vfetk_GuessType gues;  
00191     Vfetk_PrecType lprec;  
00192     int pjac;  
00194     PBEparm *pbeparm;  
00195     FEMparm *feparm;  
00196     Vhal_PBEType type;  
00197     int level;  
00199 };  
00200  
00204 typedef struct svfetk Vfetk;  
00205  
00212 struct svfetk_LocalVar {  
00213     double nvec[VAPBS_DIM];  
00214     double vx[4][VAPBS_DIM];  
00215     double xq[VAPBS_DIM];  
00216     double U[MAXV];  
00217     double dU[MAXV][VAPBS_DIM];  
00218     double W;  
00219     double dW[VAPBS_DIM];  
00220     double d2W;  
00221     int sType;  
00222     int fType;  
00223     double diel;  
00224     double ionacc;  
00225     double A;  
00226     double F;  
00227     double B;  
00228     double DB;  
00229     double jumpDiel;  
00230     Vfetk *fetk;  
00231     Vgreen *green;  
00232     int initGreen;  
00234     SS *simp;  
00236     VV *verts[4];  
00237     int nverts;  
00238     double ionConc[MAXION];
```

```

00239     double ionQ[MAXION];
00240     double ionRadii[MAXION];
00241     double zkappa2;
00242     double zks2;
00243     double ionstr;
00244     int nion;
00245     double Fu_v;
00246     double DFu_wv;
00247     double delta;
00248     double u_D;
00249     double u_T;
00250 };
00251
00256 typedef struct svfetk_LocalVar Vfetk_LocalVar;
00257
00258 #if !defined(VINLINE_VFETK)
00259
00265     VEXTERNC Gem* Vfetk_getGem(
00266         Vfetk *thee
00267     ) ;
00268
00274     VEXTERNC AM* Vfetk_getAM(
00275         Vfetk *thee
00276     ) ;
00277
00283     VEXTERNC Vpbe* Vfetk_getVpbe(
00284         Vfetk *thee
00285     ) ;
00286
00292     VEXTERNC Vcsm* Vfetk_getVcsm(
00293         Vfetk *thee
00294     ) ;
00295
00302     VEXTERNC int Vfetk_getAtomColor(
00303         Vfetk *thee,
00304         int iatom
00305     ) ;
00306
00307 #else /* if defined(VINLINE_VFETK) */
00308 # define Vfetk_getGem(thee) ((thee)->gm)
00309 # define Vfetk_getAM(thee) ((thee)->am)
00310 # define Vfetk_getVpbe(thee) ((thee)->pbe)
00311 # define Vfetk_getVcsm(thee) ((thee)->csm)
00312 # define Vfetk_getAtomColor(thee, iatom) (Vatom_getPartID(Valist_getAtom(Vpbe_g
etValist(thee->pbe), iatom))
00313 #endif /* if !defined(VINLINE_VFETK) */
00314
00315 /* //////////////////////////////// Class Vfetk: Non-Inlineable methods (vfetk.c)
00316 // Class Vfetk: Non-Inlineable methods (vfetk.c)
00318
00328 VEXTERNC Vfetk* Vfetk_ctor(
00329     Vpbe *pbe,
00330     Vhal_PBEType type
00331 ) ;
00332
00342 VEXTERNC int Vfetk_ctor2(
00343     Vfetk *thee,

```

```
00344     Vpbe *pbe,
00345     Vhal_PBEType type
00346 );
00347
00353 VEXTERNC void Vfetk_dtor(
00354     Vfetk **thee
00355 );
00356
00362 VEXTERNC void Vfetk_dtor2(
00363     Vfetk *thee
00364 );
00365
00375 VEXTERNC double* Vfetk_getSolution(
00376     Vfetk *thee,
00377     int *length
00378 );
00379
00385 VEXTERNC void Vfetk_setParameters(
00386     Vfetk *thee,
00387     PBEParm *pbeparm,
00388     FEMparm *feparm
00389 );
00390
00409 VEXTERNC double Vfetk_energy(
00410     Vfetk *thee,
00411     int color,
00412     int nonlin
00413 );
00418
00448 VEXTERNC double Vfetk_dqmEnergy(
00449     Vfetk *thee,
00450     int color
00451 );
00455
00473 VEXTERNC double Vfetk_qfEnergy(
00474     Vfetk *thee,
00475     int color
00476 );
00478
00486 VEXTERNC unsigned long int Vfetk_memChk(
00487     Vfetk *thee
00488 );
00489
00505 VEXTERNC void Vfetk_setAtomColors(
00506     Vfetk *thee
00507 );
00508
00517 VEXTERNC void Bmat_printHB(
00518     Bmat *thee,
00519     char *fname
00520 );
00521
00527 VEXTERNC Vrc_Codes Vfetk_genCube(
00528     Vfetk *thee,
00529     double center[3],
00530     double length[3],
00531     Vfetk_MeshLoad meshType
```

```
00532     );
00533
00539 VEXTERNC Vrc_Codes Vfetk_loadMesh(
00540     Vfetk *thee,
00541     double center[3],
00542     double length[3],
00543     Vfetk_MeshLoad meshType,
00544     Vio *sock
00545     );
00546
00553 VEXTERNC PDE* Vfetk_PDE_ctor(
00554     Vfetk *fetk
00555     );
00556
00563 VEXTERNC int Vfetk_PDE_ctor2(
00564     PDE *thee,
00565     Vfetk *fetk
00566     );
00567
00574 VEXTERNC void Vfetk_PDE_dtor(
00575     PDE **thee
00576     );
00577
00584 VEXTERNC void Vfetk_PDE_dtor2(
00585     PDE *thee
00586     );
00587
00593 VEXTERNC void Vfetk_PDE_initAssemble(
00594     PDE *thee,
00595     int ip[],
00596     double rp[]
00597     );
00598
00605 VEXTERNC void Vfetk_PDE_initElement(
00606     PDE *thee,
00607     int elementType,
00608     int chart,
00611     double tvx[] [VAPBS_DIM],
00612     void *data
00613     );
00614
00620 VEXTERNC void Vfetk_PDE_initFace(
00621     PDE *thee,
00622     int faceType,
00624     int chart,
00626     double tnvec[]
00627     );
00628
00636 VEXTERNC void Vfetk_PDE_initPoint(
00637     PDE *thee,
00638     int pointType,
00639     int chart,
00641     double txq[],
00642     double tU[],
00643     double tdU[] [VAPBS_DIM]
00644     );
00645
```

```
00663 VEXTERNC void Vfetk_PDE_Fu(
00664     PDE *thee,
00665     int key,
00666     double F[]
00667 );
00668
00669
00680 VEXTERNC double Vfetk_PDE_Fu_v(
00681     PDE *thee,
00682     int key,
00683     double V[],
00684     double dV[] [VAPBS_DIM]
00685 );
00686
00687
00699 VEXTERNC double Vfetk_PDE_DFu_wv(
00700     PDE *thee,
00701     int key,
00702     double W[],
00703     double dW[] [VAPBS_DIM],
00704     double V[],
00705     double dV[] [VAPBS_DIM]
00706 );
00707
00708
00715 VEXTERNC void Vfetk_PDE_delta(
00716     PDE *thee,
00717     int type,
00718     int chart,
00719     double txq[],
00720     void *user,
00721     double F[]
00722 );
00723
00731 VEXTERNC void Vfetk_PDE_u_D(
00732     PDE *thee,
00733     int type,
00734     int chart,
00735     double txq[],
00736     double F[]
00737 );
00738
00746 VEXTERNC void Vfetk_PDE_u_T(
00747     PDE *thee,
00748     int type,
00749     int chart,
00750     double txq[],
00751     double F[]
00752 );
00753
00759 VEXTERNC void Vfetk_PDE_bisectEdge(
00760     int dim,
00761     int dimII,
00762     int edgeType,
00763     int chart[],
00765     double vx[] [VAPBS_DIM]
00766 );
00767
00773 VEXTERNC void Vfetk_PDE_mapBoundary(
00774     int dim,
```

```
00775     int dimII,
00776     int vertexType,
00777     int chart,
00778     double vx[VAPBS_DIM]
00779 );
00780
00789 VEXTERNC int Vfetk_PDE_markSimplex(
00790     int dim,
00791     int dimII,
00792     int simplexType,
00793     int faceType[VAPBS_NVS],
00794     int vertexType[VAPBS_NVS],
00795     int chart[],
00796     double vx[][VAPBS_DIM],
00797     void *simplex
00798 );
00799
00805 VEXTERNC void Vfetk_PDE_oneChart(
00806     int dim,
00807     int dimII,
00808     int objType,
00809     int chart[],
00810     double vx[][VAPBS_DIM],
00811     int dimV
00812 );
00813
00823 VEXTERNC double Vfetk_PDE_Ju(
00824     PDE *thee,
00825     int key
00826 );
00827
00835 VEXTERNC void Vfetk_externalUpdateFunction(
00836     SS **simpsons,
00838     int num
00839 );
00840
00841
00904 VEXTERNC int Vfetk_PDE_simplexBasisInit(
00905     int key,
00907     int dim,
00908     int comp,
00910     int *ndof,
00911     int dof[]
00912 );
00913
00921 VEXTERNC void Vfetk_PDE_simplexBasisForm(
00922     int key,
00924     int dim,
00925     int comp ,
00926     int pdkey,
00935     double xq[],
00936     double basis[]
00938 );
00939
00945 VEXTERNC void Vfetk_readMesh(
00946     Vfetk *thee,
00947     int skey,
```

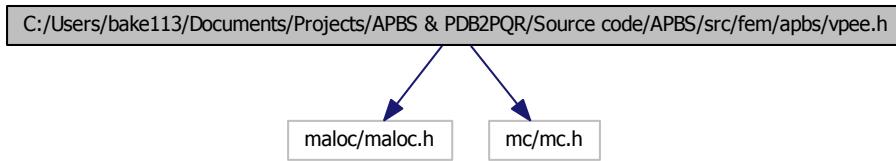
```
00948     Vio *sock
00949 );
00950
00956 VEXTERNC void Vfetk_dumpLocalVar();
00957
00965 VEXTERNC int Vfetk_fillArray(
00966     Vfetk *thee,
00967     Bvec *vec,
00968     Vdata_Type type
00969 );
00970
00985 VEXTERNC int Vfetk_write(
00986     Vfetk *thee,
00987     const char *iodev,
00989     const char *iofmt,
00991     const char *thost,
00992     const char *fname,
00993     Bvec *vec,
00994     Vdata_Format format
00995 );
00996
01002 VEXTERNC Vrc_Codes Vfetk_loadGem(
01003     Vfetk *thee,
01004     Gem *gm
01005 );
01006
01007
01008 #endif /* ifndef _VFETK_H_ */
```

10.11 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/vpee.h File Reference

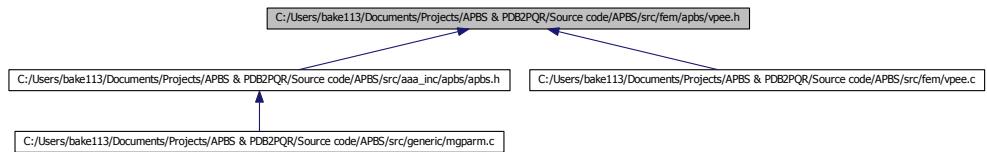
Contains declarations for class Vpee.

```
#include "maloc/maloc.h"
#include "mc/mc.h"
```

Include dependency graph for vpee.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVpee](#)

Contains public data members for Vpee class/module.

Typedefs

- typedef struct [sVpee](#) [Vpee](#)

Declaration of the Vpee class as the Vpee structure.

Functions

- VEXTERNC [Vpee](#) * [Vpee_ctor](#) (Gem *gm, int localPartID, int killFlag, double killParam)
Construct the Vpee object.
- VEXTERNC int [Vpee_ctor2](#) ([Vpee](#) *thee, Gem *gm, int localPartID, int killFlag, double killParam)
FORTRAN stub to construct the Vpee object.
- VEXTERNC void [Vpee_dtor](#) ([Vpee](#) **thee)
Object destructor.
- VEXTERNC void [Vpee_dtor2](#) ([Vpee](#) *thee)
FORTRAN stub object destructor.
- VEXTERNC int [Vpee_markRefine](#) ([Vpee](#) *thee, AM *am, int level, int akey, int rcol, double etol, int bkey)
Mark simplices for refinement based on attenuated error estimates.
- VEXTERNC int [Vpee_numSS](#) ([Vpee](#) *thee)
Returns the number of simplices in the local partition.

10.11.1 Detailed Description

Contains declarations for class Vpee.

Version

Id:

[vpee.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vpee.h](#).

10.12 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/apbs/vpee.h

```

00001
00076 #ifndef _VPEE_H
00077 #define _VPEE_H
00078
00079 /* Generic headers */
00080 #include "malloc/malloc.h"
00081 #include "mc/mc.h"
00082
00088 struct sVpee {
00089
00090     Gem *gm;
00091     int localPartID;
00094     double localPartCenter[3];
00096     double localPartRadius;
00098     int killFlag;
00101     double killParam;
00103     Vmem *mem;
00105 };
00106
00111 typedef struct sVpee Vpee;
00112
00113 /* //////////////////////////////// */
00114 // Class Vpee Inlineable methods
00116
00117 #if !defined(VINLINE_VPEE)
00118 #else /* if defined(VINLINE_VPEE) */
00119 #endif /* if !defined(VINLINE_VPEE) */
00120
00121 /* //////////////////////////////// */
00122 // Class Vpee: Non-Inlineable methods (vpee.c)
00124
00131 VEXTERNC Vpee* Vpee_ctor(
00132     Gem *gm,
00133     int localPartID,
00134     int killFlag,
00145     double killParam
00146 );
00147
00154 VEXTERNC int Vpee_ctor2(

```

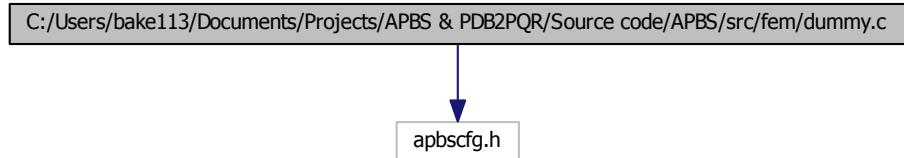
```
00155     Vpee *thee,  
00156     Gem *gm,  
00157     int localPartID,  
00158     int killFlag,  
00159     double killParam  
00170     );  
00171  
00176 VEXTERNC void Vpee_dtor(  
00177     Vpee **thee  
00178     );  
00179  
00184 VEXTERNC void Vpee_dtor2(  
00185     Vpee *thee  
00186     );  
00187  
00203 VEXTERNC int Vpee_markRefine(  
00204     Vpee *thee,  
00205     AM *am,  
00206     int level,  
00207     int akey,  
00215     int rcol,  
00218     double etol,  
00219     int bkey  
00223     );  
00224  
00230 VEXTERNC int Vpee_numSS(  
00231     Vpee *thee  
00232     );  
00233  
00234 #endif /* ifndef _VPEE_H_ */
```

10.13 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/dummy.c File Reference

Give libtool something to do.

```
#include "apbscfg.h"
```

Include dependency graph for dummy.c:



Functions

- int **APBSFEM_dummy** (int i)

10.13.1 Detailed Description

Give libtool something to do.

Author

Nathan Baker

Version

Id:

[dummy.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*
```

```
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [dummy.c](#).

10.14 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/dummy.c

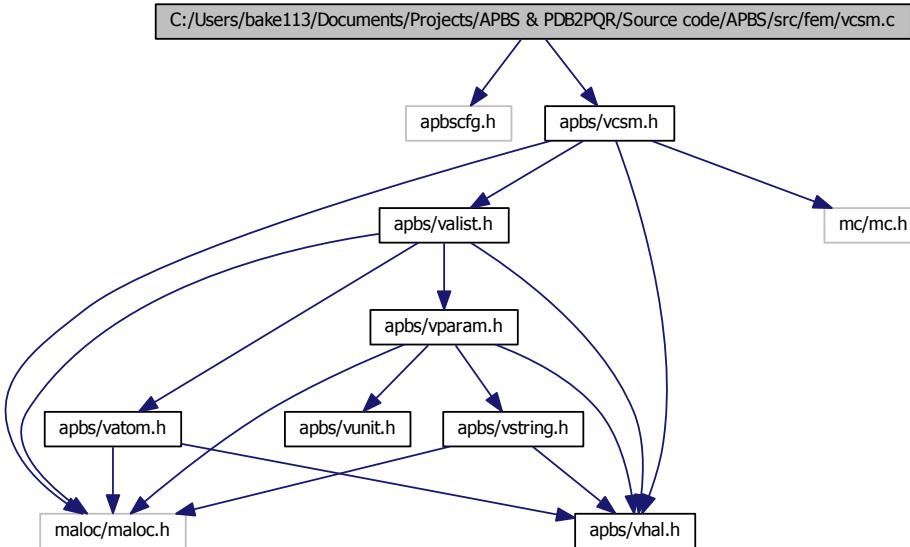
```
00001  
00056 #include "apbscfg.h"  
00057  
00058 int APBSFEM_dummy(int i) {  
00059     int j;  
00061  
00062     j = i;  
00063  
00064     return j;  
00065 }
```

10.15 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/vcsm.c File Reference

Class Vcsm methods.

```
#include "apbscfg.h"
#include "apbs/vcsm.h"
```

Include dependency graph for vcsm.c:



Functions

- VPUBLIC [Valist](#) * [Vcsm_getValist](#) ([Vcsm](#) *thee)
Get atom list.
- VPUBLIC int [Vcsm_getNumberAtoms](#) ([Vcsm](#) *thee, int isimp)
Get number of atoms associated with a simplex.
- VPUBLIC [Vatom](#) * [Vcsm_getAtom](#) ([Vcsm](#) *thee, int iatom, int isimp)
Get particular atom associated with a simplex.
- VPUBLIC int [Vcsm_getAtomIndex](#) ([Vcsm](#) *thee, int iatom, int isimp)
Get ID of particular atom in a simplex.

- VPUBLIC int `Vcsm_getNumberSimplices` (`Vcsm` *thee, int iatom)
Get number of simplices associated with an atom.
- VPUBLIC SS * `Vcsm_getSimplex` (`Vcsm` *thee, int isimp, int iatom)
Get particular simplex associated with an atom.
- VPUBLIC int `Vcsm_getSimplexIndex` (`Vcsm` *thee, int isimp, int iatom)
Get index particular simplex associated with an atom.
- VPUBLIC unsigned long int `Vcsm_memChk` (`Vcsm` *thee)
Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC `Vcsm` * `Vcsm_ctor` (`Valist` *alist, `Gem` *gm)
Construct Vcsm object.
- VPUBLIC int `Vcsm_ctor2` (`Vcsm` *thee, `Valist` *alist, `Gem` *gm)
FORTRAN stub to construct Vcsm object.
- VPUBLIC void `Vcsm_init` (`Vcsm` *thee)
Initialize charge-simplex map with mesh and atom data.
- VPUBLIC void `Vcsm_dtor` (`Vcsm` **thee)
Destroy Vcsm object.
- VPUBLIC void `Vcsm_dtor2` (`Vcsm` *thee)
FORTRAN stub to destroy Vcsm object.
- VPUBLIC int `Vcsm_update` (`Vcsm` *thee, SS **simps, int num)
Update the charge-simplex and simplex-charge maps after refinement.

10.15.1 Detailed Description

Class Vcsm methods.

Author

Nathan Baker

Version

Id:

`vcsm.c` 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*
```

```

*   Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vcsm.c](#).

10.16 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/vcsm.c

```

00001
00058 #include "apbscfg.h"
00059
00060 #if defined(HAVE_MC_H)
00061 #include "apbs/vcsm.h"
00062
00063 /* Inlineable methods */

```

```
00064 #if !defined(VINLINE_VCSTM)
00065
00066 VPUBLIC Valist* Vcsm_getValist(Vcsm *thee) {
00067     VASSERT(thee != VNULL);
00068     return thee->alist;
00069 }
00070
00071 }
00072
00073 VPUBLIC int Vcsm_getNumberAtoms(Vcsm *thee, int isimp) {
00074     VASSERT(thee != VNULL);
00075     VASSERT(thee->initFlag);
00076     return thee->nsmq[isimp];
00077 }
00078
00079 }
00080
00081 VPUBLIC Vatom* Vcsm_getAtom(Vcsm *thee, int iatom, int isimp) {
00082
00083     VASSERT(thee != VNULL);
00084     VASSERT(thee->initFlag);
00085
00086     VASSERT(iatom < (thee->nsmq)[isimp]);
00087     return Valist_getAtom(thee->alist, (thee->smq)[isimp][iatom]);
00088 }
00089
00090 }
00091
00092 VPUBLIC int Vcsm_getAtomIndex(Vcsm *thee, int iatom, int isimp) {
00093
00094     VASSERT(thee != VNULL);
00095     VASSERT(thee->initFlag);
00096
00097     VASSERT(iatom < (thee->nsmq)[isimp]);
00098     return (thee->smq)[isimp][iatom];
00099
00100 }
00101
00102
00103 VPUBLIC int Vcsm_getNumberSimplices(Vcsm *thee, int iatom) {
00104
00105     VASSERT(thee != VNULL);
00106     VASSERT(thee->initFlag);
00107
00108     return (thee->nsmq)[iatom];
00109
00110 }
00111
00112
00113 VPUBLIC SS* Vcsm_getSimplex(Vcsm *thee, int isimp, int iatom) {
00114
00115     VASSERT(thee != VNULL);
00116     VASSERT(thee->initFlag);
00117
00118     return Gem_SS(thee->gm, (thee->qsm)[iatom][isimp]);
00119
00120 }
```

```

00121 }
00122
00123 VPUBLIC int Vcsm_getSimplexIndex(Vcsm *thee, int isimp, int iatom) {
00124
00125
00126     VASSERT(thee != VNULL);
00127     VASSERT(thee->initFlag);
00128
00129     return (thee->qsm)[iatom][isimp];
00130
00131 }
00132
00133 VPUBLIC unsigned long int Vcsm_memChk(Vcsm *thee) {
00134     if (thee == VNULL) return 0;
00135     return Vmem_bytes(thee->vmem);
00136 }
00137
00138 #endif /* if !defined(VINLINE_VCSM) */
00139
00140 VPUBLIC Vcsm* Vcsm_ctor(Valist *alist, Gem *gm) {
00141
00142     /* Set up the structure */
00143     Vcsm *thee = VNULL;
00144     thee = Vmem_malloc(VNULL, 1, sizeof(Vcsm));
00145     VASSERT( thee != VNULL );
00146     VASSERT( Vcsm_ctor2(thee, alist, gm) );
00147
00148     return thee;
00149 }
00150
00151 VPUBLIC int Vcsm_ctor2(Vcsm *thee, Valist *alist, Gem *gm) {
00152
00153     VASSERT( thee != VNULL );
00154
00155     /* Memory management object */
00156     thee->vmem = Vmem_ctor("APBS:VCSM");
00157
00158     /* Set up the atom list and grid manager */
00159     if( alist == VNULL) {
00160         Vnm_print(2,"Vcsm_ctor2: got null pointer to Valist object!\n");
00161         return 0;
00162     }
00163     thee->alist = alist;
00164     if( gm == VNULL) {
00165         Vnm_print(2,"Vcsm_ctor2: got a null pointer to the Gem object!\n");
00166         return 0;
00167     }
00168     thee->gm = gm;
00169
00170     thee->initFlag = 0;
00171     return 1;
00172 }
00173
00174 VPUBLIC void Vcsm_init(Vcsm *thee) {
00175
00176     /* Counters */
00177     int iatom, jatom, isimp, jsimp, gotSimp;

```

```

00178     /* Atomic information */
00179     Vatom *atom;
00180     double *position;
00181     /* Simplex/Vertex information */
00182     SS *simplex;
00183     /* Basis function values */
00184
00185     if (thee == VNULL) {
00186         Vnm_print(2, "Vcsm_init: Error! Got NULL thee!\n");
00187         VASSERT(0);
00188     }
00189     if (thee->gm == VNULL) {
00190         VASSERT(thee->gm != VNULL);
00191         Vnm_print(2, "Vcsm_init: Error! Got NULL thee->gm!\n");
00192         VASSERT(0);
00193     }
00194     thee->nsimp = Gem_numSS(thee->gm);
00195     if (thee->nsimp <= 0) {
00196         Vnm_print(2, "Vcsm_init: Error! Got %d simplices!\n", thee->nsimp);
00197         VASSERT(0);
00198     }
00199     thee->natom = Valist_getNumberAtoms(thee->alist);
00200
00201     /* Allocate and initialize space for the first dimensions of the
00202      * simplex-charge map, the simplex array, and the counters */
00203     thee->sqm = Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int *));
00204     VASSERT(thee->sqm != VNULL);
00205     thee->nsqm = Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int));
00206     VASSERT(thee->nsqm != VNULL);
00207     for (isimp=0; isimp<thee->nsimp; isimp++) (thee->nsqm)[isimp] = 0;
00208
00209     /* Count the number of charges per simplex. */
00210     for (iatom=0; iatom<thee->natom; iatom++) {
00211         atom = Valist_getAtom(thee->alist, iatom);
00212         position = VatomGetPosition(atom);
00213         gotSimp = 0;
00214         for (isimp=0; isimp<thee->nsimp; isimp++) {
00215             simplex = Gem_SS(thee->gm, isimp);
00216             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00217                 (thee->nsqm)[isimp]++;
00218                 gotSimp = 1;
00219             }
00220         }
00221     }
00222
00223     /* Allocate the space for the simplex-charge map */
00224     for (isimp=0; isimp<thee->nsimp; isimp++) {
00225         if ((thee->nsqm)[isimp] > 0) {
00226             thee->sqm[isimp] = Vmem_malloc(thee->vmem, (thee->nsqm)[isimp],
00227                                             sizeof(int));
00228             VASSERT(thee->sqm[isimp] != VNULL);
00229         }
00230     }
00231
00232     /* Finally, set up the map */
00233     for (isimp=0; isimp<thee->nsimp; isimp++) {
00234         jsimp = 0;

```

```

00235     simplex = Gem_SS(thee->gm, isimp);
00236     for (iatom=0; iatom<thee->natom; iatom++) {
00237         atom = Valist_getAtom(thee->alist, iatom);
00238         position = Vatom_getPosition(atom);
00239         /* Check to see if the atom's in this simplex */
00240         if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00241             /* Assign the entries in the next vacant spot */
00242             (thee->sqm)[isimp][jsimp] = iatom;
00243             jsimp++;
00244         }
00245     }
00246 }
00247
00248 thee->msimp = thee->nsimp;
00249
00250 /* Allocate space for the charge-simplex map */
00251 thee->qsm = Vmem_malloc(thee->vmem, thee->natom, sizeof(int *));
00252 VASSERT(thee->qsm != VNULL);
00253 thee->nqsm = Vmem_malloc(thee->vmem, thee->natom, sizeof(int));
00254 VASSERT(thee->nqsm != VNULL);
00255 for (iatom=0; iatom<thee->natom; iatom++) (thee->nqsm)[iatom] = 0;
00256 /* Loop through the list of simplices and count the number of times
00257 * each atom appears */
00258 for (isimp=0; isimp<thee->nsimp; isimp++) {
00259     for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
00260         jatom = thee->sqm[isimp][iatom];
00261         thee->nqsm[jatom]++;
00262     }
00263 }
00264 /* Do a TIME-CONSUMING SANITY CHECK to make sure that each atom was
00265 * placed in at simplex */
00266 for (iatom=0; iatom<thee->natom; iatom++) {
00267     if (thee->nqsm[iatom] == 0) {
00268         Vnm_print(2, "Vcsm_init: Atom %d not placed in simplex!\n", iatom);
00269         VASSERT(0);
00270     }
00271 }
00272 /* Allocate the appropriate amount of space for each entry in the
00273 * charge-simplex map and clear the counter for re-use in assignment */
00274 for (iatom=0; iatom<thee->natom; iatom++) {
00275     thee->qsm[iatom] = Vmem_malloc(thee->vmem, (thee->nqsm)[iatom],
00276                                     sizeof(int));
00277     VASSERT(thee->qsm[iatom] != VNULL);
00278     thee->nqsm[iatom] = 0;
00279 }
00280 /* Assign the simplices to atoms */
00281 for (isimp=0; isimp<thee->nsimp; isimp++) {
00282     for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
00283         jatom = thee->sqm[isimp][iatom];
00284         thee->qsm[jatom][thee->nqsm[jatom]] = isimp;
00285         thee->nqsm[jatom]++;
00286     }
00287 }
00288
00289 thee->initFlag = 1;
00290 }
00291

```

```

00292 VPUBLIC void Vcsm_dtor(Vcsm **thee) {
00293     if ((*thee) != VNULL) {
00294         Vcsm_dtor2(*thee);
00295         Vmem_free(VNULL, 1, sizeof(Vcsm), (void **)thee);
00296         (*thee) = VNULL;
00297     }
00298 }
00299
00300 VPUBLIC void Vcsm_dtor2(Vcsm *thee) {
00301     int i;
00302
00303     if ((thee != VNULL) && thee->initFlag) {
00304
00305         for (i=0; i<thee->msimp; i++) {
00306             if (thee->nqsm[i] > 0) Vmem_free(thee->vmem, thee->nqsm[i],
00307                 sizeof(int), (void **)&(thee->qsm[i]));
00308         }
00309         for (i=0; i<thee->natom; i++) {
00310             if (thee->nqsm[i] > 0) Vmem_free(thee->vmem, thee->nqsm[i],
00311                 sizeof(int), (void **)&(thee->qsm[i]));
00312         }
00313         Vmem_free(thee->vmem, thee->msimp, sizeof(int *),
00314             (void **)&(thee->sqm));
00315         Vmem_free(thee->vmem, thee->msimp, sizeof(int),
00316             (void **)&(thee->nqsm));
00317         Vmem_free(thee->vmem, thee->natom, sizeof(int *),
00318             (void **)&(thee->qsm));
00319         Vmem_free(thee->vmem, thee->natom, sizeof(int),
00320             (void **)&(thee->nqsm));
00321
00322     }
00323     Vmem_dtor(&(thee->vmem));
00324 }
00325
00326 VPUBLIC int Vcsm_update(Vcsm *thee, SS **simpes, int num) {
00327
00328     /* Counters */
00329     int isimp, jsimp, iatom, jatom, atomID, simpID;
00330     int nsimps, gotMem;
00331     /* Object info */
00332     Vatom *atom;
00333     SS *simplex;
00334     double *position;
00335     /* Lists */
00336     int *qParent, nqParent;
00337     int **sqmNew, *nqsmNew;
00338     int *affAtoms, nAffAtoms;
00339     int *dnqsm, *nqsmNew, **qsmNew;
00340
00341     VASSERT(thee != VNULL);
00342     VASSERT(thee->initFlag);
00343
00344     /* If we don't have enough memory to accommodate the new entries,
00345      * add more by doubling the existing amount */
00346     isimp = thee->nsimp + num - 1;
00347     gotMem = 0;
00348     while (!gotMem) {

```

```

00349     if (isimp > thee->msimp) {
00350         isimp = 2 * isimp;
00351         thee->nsqm = Vmem_realloc(thee->vmem, thee->msimp, sizeof(int),
00352                                     (void **) &(thee->nsqm), isimp);
00353         VASSERT(thee->nsqm != VNULL);
00354         thee->sqm = Vmem_realloc(thee->vmem, thee->msimp, sizeof(int *),
00355                                     (void **) &(thee->sqm), isimp);
00356         VASSERT(thee->sqm != VNULL);
00357         thee->msimp = isimp;
00358     } else gotMem = 1;
00359 }
00360 /* Initialize the nsqm entires we just allocated */
00361 for (isimp = thee->nsimp; isimp<thee->nsimp+num-1 ; isimp++) {
00362     thee->nsqm[isimp] = 0;
00363 }
00364
00365 thee->nsimp = thee->nsimp + num - 1;
00366
00367 /* There's a simple case to deal with: if simps[0] didn't have a
00368  * charge in the first place */
00369 isimp = SS_id(simps[0]);
00370 if (thee->nsqm[isimp] == 0) {
00371     for (isimp=1; isimp<num; isimp++) {
00372         thee->nsqm[SS_id(simps[isimp])] = 0;
00373     }
00374     return 1;
00375 }
00376
00377 /* The more complicated case has occurred; the parent simplex had one or
00378  * more charges. First, generate the list of affected charges. */
00379 isimp = SS_id(simps[0]);
00380 nqParent = thee->nsqm[isimp];
00381 qParent = thee->sqm[isimp];
00382
00383 sqmNew = Vmem_malloc(thee->vmem, num, sizeof(int *));
00384 VASSERT(sqmNew != VNULL);
00385 nsqmNew = Vmem_malloc(thee->vmem, num, sizeof(int));
00386 VASSERT(nsqmNew != VNULL);
00387 for (isimp=0; isimp<num; isimp++) nsqmNew[isimp] = 0;
00388
00389 /* Loop through the affected atoms to determine how many atoms each
00390  * simplex will get. */
00391 for (iatom=0; iatom<nqParent; iatom++) {
00392
00393     atomID = qParent[iatom];
00394     atom = Valist_getAtom(thee->alist, atomID);
00395     position = Vatom_getPosition(atom);
00396     nsimps = 0;
00397
00398     jsimp = 0;
00399
00400     for (isimp=0; isimp<num; isimp++) {
00401         simplex = simps[isimp];
00402         if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00403             nsqmNew[isimp]++;
00404             jsimp = 1;
00405         }
00406     }
00407 }
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
```

```

00406         }
00407
00408         VASSERT(jsimp != 0);
00409     }
00410
00411     /* Sanity check that we didn't lose any atoms... */
00412     iatom = 0;
00413     for (isimp=0; isimp<num; isimp++) iatom += nsqmNew[isimp];
00414     if (iatom < nqParent) {
00415         Vnm_print(2,"Vcsm_update: Lost %d (of %d) atoms!\n",
00416                 nqParent - iatom, nqParent);
00417         VASSERT(0);
00418     }
00419
00420     /* Allocate the storage */
00421     for (isimp=0; isimp<num; isimp++) {
00422         if (nsqmNew[isimp] > 0) {
00423             sqmNew[isimp] = Vmem_malloc(thee->vmem, nsqmNew[isimp],
00424                                         sizeof(int));
00425             VASSERT(sqmNew[isimp] != VNULL);
00426         }
00427     }
00428
00429     /* Assign charges to simplices */
00430     for (isimp=0; isimp<num; isimp++) {
00431
00432         jsimp = 0;
00433         simplex = simps[isimp];
00434
00435         /* Loop over the atoms associated with the parent simplex */
00436         for (iatom=0; iatom<nqParent; iatom++) {
00437
00438             atomID = qParent[iatom];
00439             atom = Valist_getAtom(thee->alist, atomID);
00440             position = Vatom_getPosition(atom);
00441             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00442                 sqmNew[isimp][jsimp] = atomID;
00443                 jsimp++;
00444             }
00445         }
00446     }
00447
00448     /* Update the QSM map using the old and new SQM lists */
00449     /* The affected atoms are those contained in the parent simplex; i.e.
00450      * thee->sqm[SS_id(simps[0])] */
00451     affAtoms = thee->sqm[SS_id(simps[0])];
00452     nAffAtoms = thee->nsqm[SS_id(simps[0])];
00453
00454     /* Each of these atoms will go somewhere else; i.e., the entries in
00455      * thee->qsm are never destroyed and thee->nqsm never decreases.
00456      * However, it is possible that a subdivision could cause an atom to be
00457      * shared by two child simplices. Here we record the change, if any,
00458      * in the number of simplices associated with each atom. */
00459     dnqsm = Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00460     VASSERT(dnqsm != VNULL);
00461     nqsmNew = Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00462     VASSERT(nqsmNew != VNULL);
00463     qsmNew = Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int*));

```

```

00463     VASSERT(qsmNew != VNULL);
00464     for (iatom=0; iatom<nAffAtoms; iatom++) {
00465         dnqsm[iatom] = -1;
00466         atomID = affAtoms[iatom];
00467         for (isimp=0; isimp<num; isimp++) {
00468             for (jatom=0; jatom<nsqmNew[isimp]; jatom++) {
00469                 if (sqmNew[isimp][jatom] == atomID) dnqsm[iatom]++;
00470             }
00471         }
00472         VASSERT(dnqsm[iatom] > -1);
00473     }
00474     /* Setup the new entries in the array */
00475     for (iatom=0; iatom<nAffAtoms; iatom++) {
00476         atomID = affAtoms[iatom];
00477         qsmNew[iatom] = Vmem_malloc(thee->vmem,
00478             (dnqsm[iatom] + thee->nqsm[atomID]),
00479             sizeof(int));
00480         nqsmNew[iatom] = 0;
00481         VASSERT(qsmNew[iatom] != VNULL);
00482     }
00483     /* Fill the new entries in the array */
00484     /* First, do the modified entries */
00485     for (isimp=0; isimp<num; isimp++) {
00486         simpID = SS_id(simps[isimp]);
00487         for (iatom=0; iatom<nsqmNew[isimp]; iatom++) {
00488             atomID = sqmNew[isimp][iatom];
00489             for (jatom=0; jatom<nAffAtoms; jatom++) {
00490                 if (atomID == affAtoms[jatom]) break;
00491             }
00492             if (jatom < nAffAtoms) {
00493                 qsmNew[jatom][nqsmNew[jatom]] = simpID;
00494                 nqsmNew[jatom]++;
00495             }
00496         }
00497     }
00498     /* Now do the unmodified entries */
00499     for (iatom=0; iatom<nAffAtoms; iatom++) {
00500         atomID = affAtoms[iatom];
00501         for (isimp=0; isimp<thee->nqsm[atomID]; isimp++) {
00502             for (jsimp=0; jsimp<num; jsimp++) {
00503                 simpID = SS_id(simps[jsimp]);
00504                 if (thee->qsm[atomID][isimp] == simpID) break;
00505             }
00506             if (jsimp == num) {
00507                 qsmNew[iatom][nqsmNew[iatom]] = thee->qsm[atomID][isimp];
00508                 nqsmNew[iatom]++;
00509             }
00510         }
00511     }
00512     /* Replace the existing entries in the table. Do the QSM entires
00513      * first, since they require affAtoms = thee->sqm[simps[0]] */
00514     for (iatom=0; iatom<nAffAtoms; iatom++) {
00515         atomID = affAtoms[iatom];
00516         Vmem_free(thee->vmem, thee->nqsm[atomID], sizeof(int),
00517             (void **)&(thee->qsm[atomID]));
00518         thee->qsm[atomID] = qsmNew[iatom];
00519

```

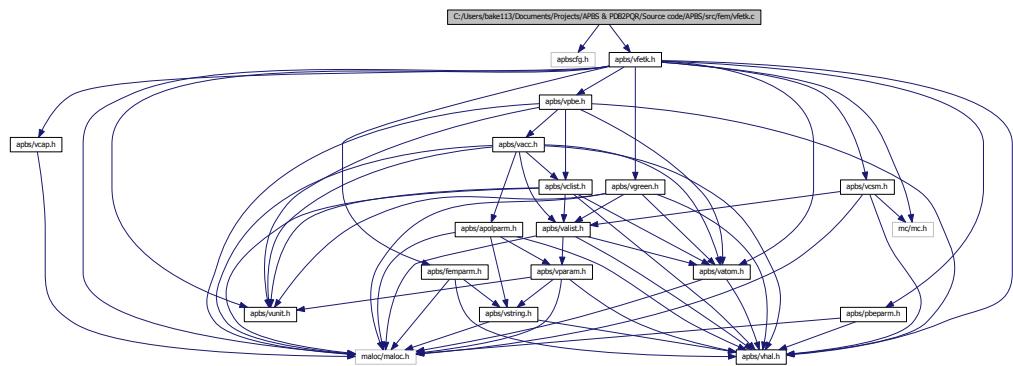
```
00520     thee->nqsm[atomID] = nqsmNew[iatom];
00521 }
00522 for (isimp=0; isimp<num; isimp++) {
00523     simpID = SS_id(simps[isimp]);
00524     if (thee->nsqm[simpID] > 0) Vmem_free(thee->vmem, thee->nsqm[simpID],
00525         sizeof(int), (void **)&(thee->sqm[simpID]));
00526     thee->sqm[simpID] = sqmNew[isimp];
00527     thee->nsqm[simpID] = nsqmNew[isimp];
00528 }
00529
00530 Vmem_free(thee->vmem, num, sizeof(int *), (void **)&sqmNew);
00531 Vmem_free(thee->vmem, num, sizeof(int), (void **)&nsqmNew);
00532 Vmem_free(thee->vmem, nAffAtoms, sizeof(int *), (void **)&qsmNew);
00533 Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&nqsmNew);
00534 Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&dnqsm);
00535
00536
00537 return 1;
00538
00539
00540 }
00541
00542 #endif
```

10.17 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source
code/APBS/src/fem/vfetk.c File Reference

Class Vfetk methods.

```
#include "apbscfg.h"  
#include "apbs/vfetk.h"
```

Include dependency graph for vfetk.c:



Defines

- #define **VMAXLOCALCOLORSDONTREUSETHISVARIABLE** 1024
- #define **VRINGMAX** 1000

Maximum number of simplices in a simplex ring.
- #define **VATOMMAX** 1000000

Maximum number of atoms associated with a vertex.

Functions

- VPRIvATE double **Vfetk_qfEnergyAtom** (**Vfetk** *thee, int iatom, int color, double *sol)
- VPRIvATE double **diel** ()
- VPRIvATE double **ionacc** ()
- VPRIvATE double **smooth** (int nverts, double dist[VAPBS_NVS], double coeff[VAPBS_-NVS], int meth)
- VPRIvATE double **debye_U** (**Vpbe** *pbe, int d, double x[])
- VPRIvATE double **debye_Udiff** (**Vpbe** *pbe, int d, double x[])
- VPRIvATE void **coulomb** (**Vpbe** *pbe, int d, double x[], double eps, double *U, double dU[], double *d2U)
- VPRIvATE void **init_2DP1** (int dimIS[], int *ndof, int dof[], double c[][VMAXP], double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP])
- VPRIvATE void **init_3DP1** (int dimIS[], int *ndof, int dof[], double c[][VMAXP], double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP])
- VPRIvATE void **setCoef** (int numP, double c[][VMAXP], double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP], int ic[], int icx[], int icy[], int icz[])

Get a pointer to the Gem (grid manager) object.
- VPUBLIC Gem * **Vfetk_getGem** (**Vfetk** *thee)

Get a pointer to the Gem (grid manager) object.
- VPUBLIC AM * **Vfetk_getAM** (**Vfetk** *thee)

Get a pointer to the AM (algebra manager) object.
- VPUBLIC **Vpbe** * **Vfetk_getVpbe** (**Vfetk** *thee)

Get a pointer to the Vpbe (PBE manager) object.
- VPUBLIC **Vcsm** * **Vfetk_getVcsm** (**Vfetk** *thee)

Get a pointer to the Vcsm (charge-simplex map) object.
- VPUBLIC int **Vfetk_getAtomColor** (**Vfetk** *thee, int iatom)

Get the partition information for a particular atom.
- VPUBLIC **Vfetk** * **Vfetk_ctor** (**Vpbe** *pbe, **Vhal_PBEType** type)

Constructor for Vfetk object.
- VPUBLIC int **Vfetk_ctor2** (**Vfetk** *thee, **Vpbe** *pbe, **Vhal_PBEType** type)

FORTRAN stub constructor for Vfetk object.

- VPUBLIC void [Vfetk_setParameters](#) ([Vfetk](#) *thee, [PBEparm](#) *pbeparm, [FEM-parm](#) *feparm)

Set the parameter objects.
- VPUBLIC void [Vfetk_dtor](#) ([Vfetk](#) **thee)

Object destructor.
- VPUBLIC void [Vfetk_dtor2](#) ([Vfetk](#) *thee)

FORTRAN stub object destructor.
- VPUBLIC double * [Vfetk_getSolution](#) ([Vfetk](#) *thee, int *length)

Create an array containing the solution (electrostatic potential in units of $k_B T / e$) at the finest mesh level.
- VPUBLIC double [Vfetk_energy](#) ([Vfetk](#) *thee, int color, int nonlin)

Return the total electrostatic energy.
- VPUBLIC double [Vfetk_qfEnergy](#) ([Vfetk](#) *thee, int color)

Get the "fixed charge" contribution to the electrostatic energy.
- VPUBLIC double [Vfetk_dqmEnergy](#) ([Vfetk](#) *thee, int color)

Get the "mobile charge" and "polarization" contributions to the electrostatic energy.
- VPUBLIC void [Vfetk_setAtomColors](#) ([Vfetk](#) *thee)

Transfer color (partition ID) information from a partitioned mesh to the atoms.
- VPUBLIC unsigned long int [Vfetk_memChk](#) ([Vfetk](#) *thee)

Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC Vrc_Codes [Vfetk_genCube](#) ([Vfetk](#) *thee, double center[3], double length[3], [Vfetk_MeshLoad](#) meshType)

Construct a rectangular mesh (in the current Vfetk object)
- VPUBLIC Vrc_Codes [Vfetk_loadMesh](#) ([Vfetk](#) *thee, double center[3], double length[3], [Vfetk_MeshLoad](#) meshType, Vio *sock)

Loads a mesh into the Vfetk (and associated) object(s).
- VPUBLIC void [Bmat_printHB](#) (Bmat *thee, char *fname)

Writes a Bmat to disk in Harwell-Boeing sparse matrix format.
- VPUBLIC PDE * [Vfetk_PDE_ctor](#) ([Vfetk](#) *fetk)

Constructs the FEtk PDE object.
- VPUBLIC int [Vfetk_PDE_ctor2](#) (PDE *thee, [Vfetk](#) *fetk)

Initializes the FEtk PDE object.
- VPUBLIC void [Vfetk_PDE_dtor](#) (PDE **thee)

Destroys FEtk PDE object.
- VPUBLIC void [Vfetk_PDE_dtor2](#) (PDE *thee)

FORTRAN stub: destroys FEtk PDE object.
- VPUBLIC void [Vfetk_PDE_initAssemble](#) (PDE *thee, int ip[], double rp[])

Do once-per-assembly initialization.
- VPUBLIC void [Vfetk_PDE_initElement](#) (PDE *thee, int elementType, int chart, double tvx[][3], void *data)

- VPUBLIC void [Vfetk_PDE_initFace](#) (PDE *thee, int faceType, int chart, double tnvec[])

Do once-per-face initialization.
- VPUBLIC void [Vfetk_PDE_initPoint](#) (PDE *thee, int pointType, int chart, double txq[], double tU[], double tDU[][3])
- VPUBLIC void [Vfetk_PDE_Fu](#) (PDE *thee, int key, double F[])

Evaluate strong form of PBE. For interior points, this is:

$$-\nabla \cdot \epsilon \nabla u + b(u) - f$$

where $b(u)$ is the (possibly nonlinear) mobile ion term and f is the source charge distribution term (for PBE) or the induced surface charge distribution (for RPBE). For an interior-boundary (simplex face) point, this is:

$$[\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^+} - [\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^-}$$

where $n(x)$ is the normal to the simplex face and the term represents the jump in dielectric displacement across the face. There is no outer-boundary contribution for this problem.

- VPUBLIC double [Vfetk_PDE_Fu_v](#) (PDE *thee, int key, double V[], double dV[][VAPBS_DIM])

This is the weak form of the PBE; i.e. the strong form integrated with a test function to give:

$$\int_{\Omega} [\epsilon \nabla u \cdot \nabla v + b(u)v - fv] dx$$

where $b(u)$ denotes the mobile ion term.

- VPUBLIC double [Vfetk_PDE_DFu_wv](#) (PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[][3])
- VPUBLIC void [Vfetk_PDE_delta](#) (PDE *thee, int type, int chart, double txq[], void *user, double F[])

Evaluate a (discretized) delta function source term at the given point.

- VPUBLIC void [Vfetk_PDE_u_D](#) (PDE *thee, int type, int chart, double txq[], double F[])

Evaluate the Dirichlet boundary condition at the given point.

- VPUBLIC void [Vfetk_PDE_u_T](#) (PDE *thee, int type, int chart, double txq[], double F[])

Evaluate the "true solution" at the given point for comparison with the numerical solution.

- VPUBLIC void [Vfetk_PDE_bisectEdge](#) (int dim, int dimll, int edgeType, int chart[], double vx[][3])
- VPUBLIC void [Vfetk_PDE_mapBoundary](#) (int dim, int dimll, int vertexType, int chart, double vx[3])
- VPUBLIC int [Vfetk_PDE_markSimplex](#) (int dim, int dimll, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][3], void *simplex)

- VPUBLIC void **Vfetk_PDE_oneChart** (int dim, int dimII, int objType, int chart[], double vx[][3], int dimV)
- VPUBLIC double **Vfetk_PDE_Ju** (PDE *thee, int key)

Energy functional. This returns the energy (less delta function terms) in the form:

$$c^{-1}/2 \int (\varepsilon(\nabla u)^2 + \kappa^2(cosh u - 1))dx$$

for a 1:1 electrolyte where c is the output from Vpbe_getZmagic.

- VPUBLIC void **Vfetk_externalUpdateFunction** (SS **simps, int num)

External hook to simplex subdivision routines in Gem. Called each time a simplex is subdivided (we use it to update the charge-simplex map)
- VPUBLIC int **Vfetk_PDE_simplexBasisInit** (int key, int dim, int comp, int *ndof, int dof[])

Initialize the bases for the trial or the test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VPUBLIC void **Vfetk_PDE_simplexBasisForm** (int key, int dim, int comp, int pdkey, double xq[], double basis[])

Evaluate the bases for the trial or test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VPUBLIC void **Vfetk_dumpLocalVar** ()

Debugging routine to print out local variables used by PDE object.
- VPUBLIC int **Vfetk_fillArray** (Vfetk *thee, Bvec *vec, **Vdata_Type** type)

Fill an array with the specified data.
- VPUBLIC int **Vfetk_write** (Vfetk *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, Bvec *vec, **Vdata_Format** format)

Write out data.

Variables

- VPRIVATE **Vfetk_LocalVar** var
- VPRIVATE char * **diriCubeString**
- VPRIVATE char * **neumCubeString**
- VPRIVATE int **dim_2DP1** = 3
- VPRIVATE int **Igr_2DP1** [3][VMAXP]
- VPRIVATE int **Igr_2DP1x** [3][VMAXP]
- VPRIVATE int **Igr_2DP1y** [3][VMAXP]
- VPRIVATE int **Igr_2DP1z** [3][VMAXP]
- VPRIVATE int **dim_3DP1** = VAPBS_NVS
- VPRIVATE int **Igr_3DP1** [VAPBS_NVS][VMAXP]
- VPRIVATE int **Igr_3DP1x** [VAPBS_NVS][VMAXP]
- VPRIVATE int **Igr_3DP1y** [VAPBS_NVS][VMAXP]
- VPRIVATE int **Igr_3DP1z** [VAPBS_NVS][VMAXP]

- VPRIVATE const int **P_DEG** = 1
- VPRIVATE int **numP**
- VPRIVATE double **c** [VMAXP][VMAXP]
- VPRIVATE double **cx** [VMAXP][VMAXP]
- VPRIVATE double **cy** [VMAXP][VMAXP]
- VPRIVATE double **cz** [VMAXP][VMAXP]

10.17.1 Detailed Description

Class Vfetk methods.

Author

Nathan Baker

Version

Id:

[vfetk.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation
```

* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

Definition in file [vfetk.c](#).

10.17.2 Variable Documentation

10.17.2.1 VPRIVATE char* diriCubeString

Initial value:

```

"mcsf_begin=1;\n\
\n\
dim=3;\n\
dimii=3;\n\
vertices=8;\n\
simplices=6;\n\
\n\
vert=[\n\
0 0 -0.5 -0.5 -0.5\n\
1 0 0.5 -0.5 -0.5\n\
2 0 -0.5 0.5 -0.5\n\
3 0 0.5 0.5 -0.5\n\
4 0 -0.5 -0.5 0.5\n\
5 0 0.5 -0.5 0.5\n\
6 0 -0.5 0.5 0.5\n\
7 0 0.5 0.5 0.5\n\
];\n\
\n\
simp=[\n\
0 0 0 0 1 0 1 0 5 1 2\n\
1 0 0 0 1 1 0 0 5 2 4\n\
2 0 0 0 1 0 1 1 5 3 2\n\
3 0 0 0 1 0 1 3 5 7 2\n\
4 0 0 1 1 0 0 2 5 7 6\n\
5 0 0 1 1 0 0 2 5 6 4\n\

```

```
];\n\
\n\
mcsf_end=1;\n\
\n\
"
```

Definition at line 98 of file [vfetk.c](#).

10.17.2.2 VPRIVATE int lgr_2DP1[3][VMAXP]

Initial value:

```
{
{ 2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 391 of file [vfetk.c](#).

10.17.2.3 VPRIVATE int lgr_2DP1x[3][VMAXP]

Initial value:

```
{
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 400 of file [vfetk.c](#).

10.17.2.4 VPRIVATE int lgr_2DP1y[3][VMAXP]

Initial value:

```
{
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 407 of file [vfetk.c](#).

10.17.2.5 VPRIVATE int lgr_2DP1z[3][VMAXP]

Initial value:

```
{  
  
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }  
}
```

Definition at line 414 of file vfetk.c.

10.17.2.6 VPRIVATE int lgr_3DP1[VAPBS_NVS][VMAXP]

Initial value:

```
{  
  
{ 2, -2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
{ 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
{ 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
{ 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }  
}
```

Definition at line 443 of file vfetk.c.

10.17.2.7 VPRIVATE int lgr_3DP1x[VAPBS_NVS][VMAXP]

Initial value:

```
{  
  
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }  
}
```

Definition at line 451 of file vfetk.c.

10.17.2.8 VPRIVATE int lgr_3DP1y[VAPBS_NVS][VMAXP]

Initial value:

```
{
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 459 of file [vfetk.c](#).

10.17.2.9 VPRIVATE int lgr_3DP1z[VAPBS_NVS][VMAXP]

Initial value:

```
{
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 467 of file [vfetk.c](#).

10.17.2.10 VPRIVATE char* neumCubeString

Initial value:

```
"mcsf_begin=1;\n\
\n\
dim=3;\n\
dimmi=3;\n\
vertices=8;\n\
simplices=6;\n\
\n\
vert=[\n\
0 0 -0.5 -0.5 -0.5\n\
1 0 0.5 -0.5 -0.5\n\
2 0 -0.5 0.5 -0.5\n\
3 0 0.5 0.5 -0.5\n\
4 0 -0.5 -0.5 0.5\n\
5 0 0.5 -0.5 0.5\n\
6 0 -0.5 0.5 0.5\n\
7 0 0.5 0.5 0.5\n\
];\n\
\n\
simp=[\n\
0 0 0 2 0 2 0 5 1 2\n\
1 0 0 0 2 2 0 0 5 2 4\n\
]
```

```
2 0 0 0 2 0 2 1 5 3 2\n\
3 0 0 0 2 0 2 3 5 7 2\n\
4 0 0 2 2 0 0 2 5 7 6\n\
5 0 0 2 2 0 0 2 5 6 4\n\
];\n\
\n\
mcsf_end=1;\n\
\n\
"
```

Definition at line 135 of file vfetk.c.

10.18 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/vfetk.c

```
00001
00058 #include "apbscfg.h"
00059
00060 #ifdef HAVE_MC_H
00061
00062 #include "apbs/vfetk.h"
00063
00064 /* Define the macro DONEUMANN to run with all-Neumann boundary conditions.
00065 * Set this macro at your own risk! */
00066 /* #define DONEUMANN 1 */
00067
00068 /*
00069 * @brief Calculate the contribution to the charge-potential energy from one
00070 * atom
00071 * @ingroup Vfetk
00072 * @author Nathan Baker
00073 * @param thee current Vfetk object
00074 * @param iatom current atom index
00075 * @param color simplex subset (partition) under consideration
00076 * @param sol current solution
00077 * @returns Per-atom energy
00078 */
00079 VPRIIVATE double Vfetk_qfEnergyAtom(
00080         Vfetk *thee,
00081         int iatom,
00082         int color,
00083         double *sol
00084     );
00085
00086 /*
00087 * @brief Container for local variables
00088 * @ingroup Vfetk
00089 * @bug Not thread-safe
00090 */
00091 VPRIIVATE Vfetk_LocalVar var;
00092
00093 /*
00094 * @brief MCSF-format cube mesh (all Dirichlet)
```

```

00095 * @ingroup Vfetk
00096 * @author Based on mesh by Mike Holst
00097 */
00098 VPRIIVATE char *diriCubeString =
00099 "mcsf_begin=1;\n\
00100 \n\
00101 dim=3;\n\
00102 dimii=3;\n\
00103 vertices=8;\n\
00104 simplices=6;\n\
00105 \n\
00106 vert=[\n\
00107 0 0 -0.5 -0.5 -0.5\n\
00108 1 0 0.5 -0.5 -0.5\n\
00109 2 0 -0.5 0.5 -0.5\n\
00110 3 0 0.5 0.5 -0.5\n\
00111 4 0 -0.5 -0.5 0.5\n\
00112 5 0 0.5 -0.5 0.5\n\
00113 6 0 -0.5 0.5 0.5\n\
00114 7 0 0.5 0.5 0.5\n\
00115 ];\n\
00116 \n\
00117 simp=[\n\
00118 0 0 0 0 1 0 1 0 5 1 2\n\
00119 1 0 0 0 1 1 0 0 5 2 4\n\
00120 2 0 0 0 1 0 1 1 5 3 2\n\
00121 3 0 0 0 1 0 1 3 5 7 2\n\
00122 4 0 0 1 1 0 0 2 5 7 6\n\
00123 5 0 0 1 1 0 0 2 5 6 4\n\
00124 ];\n\
00125 \n\
00126 mcsf_end=1;\n\
00127 \n\
00128 ";
00129
00130 /*
00131 * @brief MCSF-format cube mesh (all Neumann)
00132 * @ingroup Vfetk
00133 * @author Based on mesh by Mike Holst
00134 */
00135 VPRIIVATE char *neumCubeString =
00136 "mcsf_begin=1;\n\
00137 \n\
00138 dim=3;\n\
00139 dimii=3;\n\
00140 vertices=8;\n\
00141 simplices=6;\n\
00142 \n\
00143 vert=[\n\
00144 0 0 -0.5 -0.5 -0.5\n\
00145 1 0 0.5 -0.5 -0.5\n\
00146 2 0 -0.5 0.5 -0.5\n\
00147 3 0 0.5 0.5 -0.5\n\
00148 4 0 -0.5 -0.5 0.5\n\
00149 5 0 0.5 -0.5 0.5\n\
00150 6 0 -0.5 0.5 0.5\n\
00151 7 0 0.5 0.5 0.5\n\

```

```

00152  ];\n\
00153  \n\
00154  simp=[\n\
00155  0 0 0 0 2 0 2 0 5 1 2\n\
00156  1 0 0 0 2 2 0 0 5 2 4\n\
00157  2 0 0 0 2 0 2 1 5 3 2\n\
00158  3 0 0 0 2 0 2 3 5 7 2\n\
00159  4 0 0 2 2 0 0 2 5 7 6\n\
00160  5 0 0 2 2 0 0 2 5 6 4\n\
00161  ];\n\
00162  \n\
00163  mcsf_end=1;\n\
00164  \n\
00165  ";
00166
00167  /*
00168  * @brief  Return the smoothed value of the dielectric coefficient at the
00169  * current point using a fast, chart-based method
00170  * @ingroup Vfetk
00171  * @author Nathan Baker
00172  * @returns Value of dielectric coefficient
00173  * @bug Not thread-safe
00174  */
00175 VPRIVATE double diel();
00176
00177 /*
00178  * @brief  Return the smoothed value of the ion accessibility at the
00179  * current point using a fast, chart-based method
00180  * @ingroup Vfetk
00181  * @author Nathan Baker
00182  * @returns Value of mobile ion coefficient
00183  * @bug Not thread-safe
00184  */
00185 VPRIVATE double ionacc();
00186
00187 /*
00188  * @brief  Smooths a mesh-based coefficient with a simple harmonic function
00189  * @ingroup Vfetk
00190  * @author Nathan Baker
00191  * @param meth Method for smoothing
00192  *   \li 0 ==> arithmetic mean (gives bad results)
00193  *   \li 1 ==> geometric mean
00194  * @param nverts Number of vertices
00195  * @param dist distance from point to each vertex
00196  * @param coeff coefficient value at each vertex
00197  * @note Thread-safe
00198  * @return smoothed value of coefficient at point of interest */
00199 VPRIVATE double smooth(
00200         int nverts,
00201         double dist[VAPBS_NVS],
00202         double coeff[VAPBS_NVS],
00203         int meth
00204     );
00205
00206
00207 /*
00208  * @brief  Return the analytical multi-sphere Debye-Huckel approximation (in

```

```

00209 * kT/e) at the specified point
00210 * @ingroup Vfetk
00211 * @author Nathan Baker
00212 * @param pbe Vpbe object
00213 * @param d Dimension of x
00214 * @param x Coordinates of point of interest (in &Aring;)
00215 * @note Thread-safe
00216 * @returns Multi-sphere Debye-Huckel potential in kT/e
00217 */
00218 VPRIIVATE double debye_U(
00219     Vpbe *pbe,
00220     int d,
00221     double x[]
00222 );
00223
00224 /*
00225 * @brief Return the difference between the analytical multi-sphere
00226 * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
00227 * point
00228 * @ingroup Vfetk
00229 * @author Nathan Baker
00230 * @param pbe Vpbe object
00231 * @param d Dimension of x
00232 * @param x Coordinates of point of interest (in &Aring;)
00233 * @note Thread-safe
00234 * @returns Multi-sphere Debye-Huckel potential in kT/e */
00235 VPRIIVATE double debye_Udiff(
00236     Vpbe *pbe,
00237     int d,
00238     double x[]
00239 );
00240
00241 /*
00242 * @brief Calculate the Coulomb's
00243 * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
00244 * point
00245 * @ingroup Vfetk
00246 * @author Nathan Baker
00247 * @param pbe Vpbe object
00248 * @param d Dimension of x
00249 * @param x Coordinates of point of interest (in &Aring;)
00250 * @param eps Dielectric constant
00251 * @param U Set to potential (in kT/e)
00252 * @param dU Set to potential gradient (in kT/e/&Aring;)
00253 * @param d2U Set to Laplacian of potential (in \f$ kT e^{-1} \AA^{-2} \f$)
00254 * @returns Multi-sphere Debye-Huckel potential in kT/e */
00255 VPRIIVATE void coulomb(
00256     Vpbe *pbe,
00257     int d,
00258     double x[],
00259     double eps,
00260     double *U,
00261     double dU[],
00262     double *d2U
00263 );
00264
00265 /*

```

```
00266 * @brief 2D linear master simplex information generator
00267 * @ingroup Vfetk
00268 * @author Mike Holst
00269 * @param dimIS dunno
00270 * @param ndof dunno
00271 * @param dof dunno
00272 * @param c dunno
00273 * @param cx dunno
00274 * @note Trust in Mike */
00275 VPRIvate void init_2DP1(
00276     int dimIS[],
00277     int *ndof,
00278     int dof[],
00279     double c[][VMAXP],
00280     double cx[][VMAXP],
00281     double cy[][VMAXP],
00282     double cz[][VMAXP]
00283 );
00284 /*
00285 /*
00286 * @brief 3D linear master simplex information generator
00287 * @ingroup Vfetk
00288 * @author Mike Holst
00289 * @param dimIS dunno
00290 * @param ndof dunno
00291 * @param dof dunno
00292 * @param c dunno
00293 * @param cx dunno
00294 * @param cy dunno
00295 * @param cz dunno
00296 * @note Trust in Mike */
00297 VPRIvate void init_3DP1(
00298     int dimIS[],
00299     int *ndof,
00300     int dof[],
00301     double c[][VMAXP],
00302     double cx[][VMAXP],
00303     double cy[][VMAXP],
00304     double cz[][VMAXP]
00305 );
00306 /*
00307 /*
00308 * @brief Setup coefficients of polynomials from integer table data
00309 * @ingroup Vfetk
00310 * @author Mike Holst
00311 * @param numP dunno
00312 * @param c dunno
00313 * @param cx dunno
00314 * @param cy dunno
00315 * @param cz dunno
00316 * @param icx dunno
00317 * @param icy dunno
00318 * @param icz dunno
00319 * @note Trust in Mike */
00320 VPRIvate void setCoef(
00321     int numP,
```

```

00323     double c[] [VMAXP],
00324     double cx[] [VMAXP],
00325     double cy[] [VMAXP],
00326     double cz[] [VMAXP],
00327     int ic[] [VMAXP],
00328     int icx[] [VMAXP],
00329     int icy[] [VMAXP],
00330     int icz[] [VMAXP]
00331   );
00332
00333 /*
00334 * @brief Evaluate a collection of at most cubic polynomials at a
00335 * specified point in at most R^3.
00336 * @ingroup Vfetk
00337 * @author Mike Holst
00338 * @param numP the number of polynomials to evaluate
00339 * @param p the results of the evaluation
00340 * @param c the coefficients of each polynomial
00341 * @param xv the point (x,y,z) to evaluate the polynomials.
00342 * @note Mike says:
00343 * <pre>
00344 * Note that "VMAXP" must be >= 19 for cubic polynomials.
00345 * The polynomials are build from the coefficients c[][] as
00346 * follows. To build polynomial "k", fix k and set:
00347 *
00348 * c0=c[k][0], c1=c[k][1], ..., cp=c[k][p]
00349 *
00350 * Then evaluate as:
00351 *
00352 * p3(x,y,z) = c0 + c1*x + c2*y + c3*z
00353 *               + c4*x*x + c5*y*y + c6*z*z + c7*x*y + c8*x*z + c9*y*z
00354 *               + c10*x*x*x + c11*y*y*y + c12*z*z*z
00355 *               + c13*x*x*x*y + c14*x*x*z + c15*x*y*y
00356 *               + c16*y*y*z + c17*x*z*z + c18*y*z*z
00357 * </pre>
00358 */
00359 VPRIIVATE void polyEval(
00360     int numP,
00361     double p[],
00362     double c[] [VMAXP],
00363     double xv[]
00364   );
00365
00366 /*
00367 * @brief I have no clue what this variable does, but we need it to initialize
00368 * the simplices
00369 * @ingroup Vfetk
00370 * @author Mike Holst */
00371 VPRIIVATE int dim_2DP1 = 3;
00372
00373 /*
00374 * @brief I have no clue what these variable do, but we need it to initialize
00375 * the simplices
00376 * @ingroup Vfetk
00377 * @author Mike Holst
00378 * @note Mike says:
00379 * <pre>

```

```

00380 * 2D-P1 Basis:
00381 *
00382 * p1(x,y) = c0 + c1*x + c2*y
00383 *
00384 * Lagrange Point      Lagrange Basis Function Definition
00385 * -----
00386 * (0, 0)                p[0](x,y) = 1 - x - y
00387 * (1, 0)                p[1](x,y) = x
00388 * (0, 1)                p[2](x,y) = y
00389 * </pre>
00390 */
00391 VPRIPRIVATE int lgr_2DP1[3][VMAXP] = {
00392 /*c0 c1 c2 c3
00393 * -----
00394 /* 1   x   y   z
00395 * -----
00396 { 2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00397 { 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00398 { 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00399 };
00400 VPRIPRIVATE int lgr_2DP1x[3][VMAXP] = {
00401 /*c0 -----
00402 /* 1 -----
00403 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00404 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00405 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00406 };
00407 VPRIPRIVATE int lgr_2DP1y[3][VMAXP] = {
00408 /*c0 -----
00409 /* 1 -----
00410 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00411 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00412 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00413 };
00414 VPRIPRIVATE int lgr_2DP1z[3][VMAXP] = {
00415 /*c0 -----
00416 /* 1 -----
00417 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00418 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00419 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00420 };
00421
00422
00423 /*
00424 * @brief I have no clue what these variable do, but we need it to initialize
00425 * the simplices
00426 * @ingroup Vfetk
00427 * @author Mike Holst
00428 * @note Mike says:
00429 * <pre>
00430 * 3D-P1 Basis:
00431 *
00432 * p1(x,y,z) = c0 + c1*x + c2*y + c3*z
00433 *
00434 * Lagrange Point      Lagrange Basis Function Definition
00435 * -----
00436 * (0, 0, 0)            p[0](x,y,z) = 1 - x - y - z

```

```

00437 * (1, 0, 0)           p[1](x,y,z) = x
00438 * (0, 1, 0)           p[2](x,y,z) = y
00439 * (0, 0, 1)           p[3](x,y,z) = z
00440 * </pre>
00441 */
00442 VPRIIVATE int dim_3DP1 = VAPBS_NVS;
00443 VPRIIVATE int lgr_3DP1[VAPBS_NVS][VMAXP] = {
00444 /*c0 c1 c2 c3 ----- */
00445 /* 1 x y z ----- */
00446 { 2, -2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00447 { 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00448 { 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00449 { 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00450 };
00451 VPRIIVATE int lgr_3DP1x[VAPBS_NVS][VMAXP] = {
00452 /*c0 ----- */
00453 /* 1 -----
00454 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00455 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00456 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00457 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00458 };
00459 VPRIIVATE int lgr_3DP1y[VAPBS_NVS][VMAXP] = {
00460 /*c0 ----- */
00461 /* 1 -----
00462 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00463 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00464 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00465 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00466 };
00467 VPRIIVATE int lgr_3DP1z[VAPBS_NVS][VMAXP] = {
00468 /*c0 ----- */
00469 /* 1 -----
00470 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00471 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00472 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00473 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00474 };
00475
00476 /*
00477 * @brief Another Holst variable
00478 * @ingroup Vfetk
00479 * @author Mike Holst
00480 * @note Mike says: 1 = linear, 2 = quadratic */
00481 VPRIIVATE const int P_DEG=1;
00482
00483 /*
00484 * @brief Another Holst variable
00485 * @ingroup Vfetk
00486 * @author Mike Holst */
00487 VPRIIVATE int numP;
00488 VPRIIVATE double c[VMAXP][VMAXP];
00489 VPRIIVATE double cx[VMAXP][VMAXP];
00490 VPRIIVATE double cy[VMAXP][VMAXP];
00491 VPRIIVATE double cz[VMAXP][VMAXP];
00492
00493 #if !defined(VINLINE_VFETK)

```

```

00494
00495 VPUBLIC Gem* Vfetk_getGem(Vfetk *thee) {
00496
00497     VASSERT(thee != VNULL);
00498     return thee->gm;
00499
00500 }
00501
00502 VPUBLIC AM* Vfetk_getAM(Vfetk *thee) {
00503
00504     VASSERT(thee != VNULL);
00505     return thee->am;
00506 }
00507
00508 VPUBLIC Vpbe* Vfetk_getVpbe(Vfetk *thee) {
00509
00510     VASSERT(thee != VNULL);
00511     return thee->pbe;
00512
00513 }
00514
00515 VPUBLIC Vcsm* Vfetk_getVcsm(Vfetk *thee) {
00516
00517     VASSERT(thee != VNULL);
00518     return thee->csm;
00519
00520 }
00521
00522 VPUBLIC int Vfetk_getAtomColor(Vfetk *thee, int iatom) {
00523
00524     int natoms;
00525
00526     VASSERT(thee != VNULL);
00527
00528     natoms = Valist_getNumberAtoms(Vpbe_getValist(thee->pbe));
00529     VASSERT(iatom < natoms);
00530
00531     return Vatom_getPartID(Valist_getAtom(Vpbe_getValist(thee->pbe), iatom));
00532 }
00533 #endif /* if !defined(VINLINE_VFETK) */
00534
00535 VPUBLIC Vfetk* Vfetk_ctor(Vpbe *pbe, Vhal_PBEType type) {
00536
00537     /* Set up the structure */
00538     Vfetk *thee = VNULL;
00539     thee = Vmem_malloc(VNULL, 1, sizeof(Vfetk) );
00540     VASSERT(thee != VNULL);
00541     VASSERT(Vfetk_ctor2(thee, pbe, type));
00542
00543     return thee;
00544 }
00545
00546 VPUBLIC int Vfetk_ctor2(Vfetk *thee, Vpbe *pbe, Vhal_PBEType type) {
00547
00548     int i;
00549     double center[VAPBS_DIM];
00550

```

```

00551     /* Make sure things have been properly initialized & store them */
00552     VASSERT(pbe != VNULL);
00553     thee->pbe = pbe;
00554     VASSERT(pbe->alist != VNULL);
00555     VASSERT(pbe->acc != VNULL);
00556
00557     /* Store PBE type */
00558     thee->type = type;
00559
00560     /* Set up memory management object */
00561     thee->vmem = Vmem_ctor("APBS::VFETK");
00562
00563     /* Set up FETk objects */
00564     Vnm_print(0, "Vfetk_ctor2: Constructing PDE...\n");
00565     thee->pde = Vfetk_PDE_ctor(thee);
00566     Vnm_print(0, "Vfetk_ctor2: Constructing Gem...\n");
00567     thee->gm = Gem_ctor(thee->vmem, thee->pde);
00568     Vnm_print(0, "Vfetk_ctor2: Constructing Aprx...\n");
00569     thee->aprx = Aprx_ctor(thee->vmem, thee->gm, thee->pde);
00570     Vnm_print(0, "Vfetk_ctor2: Constructing Aprx...\n");
00571     thee->am = AM_ctor(thee->vmem, thee->aprx);
00572
00573     /* Reset refinement level */
00574     thee->level = 0;
00575
00576     /* Set default solver variables */
00577     thee->lkey = VLT_MG;
00578     thee->lmax = 1000000;
00579     thee->ltol = 1e-5;
00580     thee->lprec = VPT_MG;
00581     thee->nkey = VNT_NEW;
00582     thee->nmax = 1000000;
00583     thee->ntol = 1e-5;
00584     thee->gues = VGT_ZERO;
00585     thee->pjac = -1;
00586
00587     /* Store local copy of myself */
00588     var.fetk = thee;
00589     var.initGreen = 0;
00590
00591     /* Set up the external Gem subdivision hook */
00592     Gem_setExternalUpdateFunction(thee->gm, Vfetk_externalUpdateFunction);
00593
00594     /* Set up ion-related variables */
00595     var.zkappa2 = Vpbe_getZkappa2(var.fetk->pbe);
00596     var.ionstr = Vpbe_getBulkIonicStrength(var.fetk->pbe);
00597     if (var.ionstr > 0.0) var.zks2 = 0.5*var.zkappa2/var.ionstr;
00598     else var.zks2 = 0.0;
00599     Vpbe_getIons(var.fetk->pbe, &(var.nion), var.ionConc, var.ionRadii,
00600         var.ionQ);
00601     for (i=0; i<var.nion; i++) {
00602         var.ionConc[i] = var.zks2 * var.ionConc[i] * var.ionQ[i];
00603     }
00604
00605     /* Set uninitialized objects to NULL */
00606     thee->pbeparm = VNULL;
00607     thee->feparm = VNULL;

```

```

00608     thee->csm = VNULL;
00609
00610     return 1;
00611 }
00612
00613 VPUBLIC void Vfetk_setParameters(Vfetk *thee, PBparm *pbparm,
00614 FEMparm *feparm) {
00615
00616     VASSERT(thee != VNULL);
00617     thee->feparm = feparm;
00618     thee->pbparm = pbparm;
00619 }
00620
00621 VPUBLIC void Vfetk_dtor(Vfetk **thee) {
00622     if ((*thee) != VNULL) {
00623         Vfetk_dtor2(*thee);
00624         //Vmem_free(VNULL, 1, sizeof(Vfetk), (void **)thee);
00625         (*thee) = VNULL;
00626     }
00627 }
00628
00629 VPUBLIC void Vfetk_dtor2(Vfetk *thee) {
00630     Vcsm_dtor(&(thee->csm));
00631     AM_dtor(&(thee->am));
00632     Aprx_dtor(&(thee->aprx));
00633     Vfetk_PDE_dtor(&(thee->pde));
00634     Vmem_dtor(&(thee->vmem));
00635 }
00636
00637 VPUBLIC double* Vfetk_getSolution(Vfetk *thee, int *length) {
00638
00639     int i;
00640     double *solution;
00641     double *theAnswer;
00642     AM *am;
00643
00644     VASSERT(thee != VNULL);
00645
00646     /* Get the AM object */
00647     am = thee->am;
00648     /* Copy the solution into the w0 vector */
00649     Bvec_copy(am->w0, am->u);
00650     /* Add the Dirichlet conditions */
00651     Bvec_axpy(am->w0, am->ud, 1.);
00652     /* Get the data from the Bvec */
00653     solution = Bvec_addr(am->w0);
00654     /* Get the length of the data from the Bvec */
00655     *length = Bvec_numRT(am->w0);
00656     /* Make sure that we got scalar data (only one block) for the solution
00657      * to the FETK */
00658     VASSERT(1 == Bvec_numB(am->w0));
00659     /* Allocate space for the returned vector and copy the solution into it */
00660     theAnswer = Vmem_malloc(VNULL, *length, sizeof(double));
00661     VASSERT(theAnswer != VNULL);
00662     VASSERT(theAnswer != VNULL);
00663     for (i=0; i<(*length); i++) theAnswer[i] = solution[i];
00664

```

```

00665     return theAnswer;
00666 }
00667
00668 VPUBLIC double Vfetk_energy(Vfetk *thee, int color, int nonlin) {
00669
00670     double totEnergy = 0.0;
00671     double qfEnergy = 0.0;
00672     double dqmEnergy = 0.0;
00673
00674     VASSERT(thee != VNULL);
00675
00676     if (nonlin && (Vpbe_getBulkIonicStrength(thee->pbe) > 0.)) {
00677         Vnm_print(0, "Vfetk_energy: calculating full PBE energy\n");
00678         Vnm_print(0, "Vfetk_energy: bulk ionic strength = %g M\n",
00679                     Vpbe_getBulkIonicStrength(thee->pbe));
00680         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00681         Vnm_print(0, "Vfetk_energy: dqmEnergy = %g kT\n", dqmEnergy);
00682         qfEnergy = Vfetk_qfEnergy(thee, color);
00683         Vnm_print(0, "Vfetk_energy: qfEnergy = %g kT\n", qfEnergy);
00684
00685         totEnergy = qfEnergy - dqmEnergy;
00686     } else {
00687         Vnm_print(0, "Vfetk_energy: calculating only q-phi energy\n");
00688         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00689         Vnm_print(0, "Vfetk_energy: dqmEnergy = %g kT (NOT USED)\n", dqmEnergy);
00690
00691         qfEnergy = Vfetk_qfEnergy(thee, color);
00692         Vnm_print(0, "Vfetk_energy: qfEnergy = %g kT\n", qfEnergy);
00693         totEnergy = 0.5*qfEnergy;
00694     }
00695
00696     return totEnergy;
00697 }
00698
00699
00700 VPUBLIC double Vfetk_qfEnergy(Vfetk *thee, int color) {
00701
00702     double *sol; int nsol;
00703     int iatom, natoms;
00704     double energy = 0.0;
00705
00706     AM *am;
00707
00708     VASSERT(thee != VNULL);
00709     am = thee->am;
00710
00711     /* Get the finest level solution */
00712     sol= VNULL;
00713     sol = Vfetk_getSolution(thee, &nsol);
00714     VASSERT(sol != VNULL);
00715
00716     /* Make sure the number of entries in the solution array matches the
00717      * number of vertices currently in the mesh */
00718     if (nsol != Gem_numVV(thee->gm)) {
00719         Vnm_print(2, "Vfetk_qfEnergy: Number of unknowns in solution does not matc
h\n");

```

```

00720     Vnm_print(2, "Vfetk_qfEnergy: number of vertices in mesh!!! Bailing out!\n");
00721     VASSERT(0);
00722 }
00723
00724 /* Now we do the sum over atoms...
00725 natoms = Valist_getNumberAtoms(thee->pbe->alist);
00726 for (iatom=0; iatom<natoms; iatom++) {
00727
00728     energy = energy + Vfetk_qfEnergyAtom(thee, iatom, color, sol);
00729
00730 } /* end for iatom */
00731
00732 /* Destroy the finest level solution */
00733 Vmem_free(VNULL, nsol, sizeof(double), (void **) &sol);
00734
00735 /* Return the energy */
00736 return energy;
00737 }
00738
00739 VPRIvate double Vfetk_qfEnergyAtom(
00740     Vfetk *thee,
00741     int iatom,
00742     int color,
00743     double *sol) {
00744
00745     Vatom *atom;
00746     double charge;
00747     double phi[VAPBS_NVS], phix[VAPBS_NVS][3], *position;
00748     double uval;
00749     double energy = 0.0;
00750     int isimp, nsimps;
00751     SS *simp;
00752     int icolor, invert, usingColor;
00753
00754
00755     /* Get atom information */
00756     atom = Valist_getAtom(thee->pbe->alist, iatom);
00757     icolor = Vfetk_getAtomColor(thee, iatom);
00758     charge = Vatom_getCharge(atom);
00759     position = Vatom_getPosition(atom);
00760
00761     /* Find out if we're using colors */
00762     usingColor = (color >= 0);
00763
00764     if (usingColor && (icolor<0)) {
00765         Vnm_print(2, "Vfetk_qfEnergy: Atom colors not set!\n");
00766         VASSERT(0);
00767     }
00768
00769     /* Check if this atom belongs to the specified partition */
00770     if ((icolor==color) || (!usingColor)) {
00771         /* Loop over the simps associated with this atom */
00772         nsimps = Vcsm_getNumberSimplices(thee->csm, iatom);
00773
00774         /* Get the first simp of the correct color; we can use just one
00775          * simplex for energy evaluations, but not for force

```

```

00776     * evaluations */
00777     for (isimp=0; isimp<nsimps; isimp++) {
00778
00779         simp = Vcsm_getSimplex(thee->csm, isimp, iatom);
00780
00781         /* If we've asked for a particular partition AND if the atom
00782         * is our partition, then compute the energy */
00783         if ((SS_chart(simp)==color)|| (color<0)) {
00784             /* Get the value of each basis function evaluated at this
00785             * point */
00786             Gem_pointInSimplexVal(thee->gm, simp, position, phi, phix);
00787             for (ivert=0; ivert<SS_dimVV(simp); ivert++) {
00788                 uval = sol[VV_id(SS_vertex(simp,ivert))];
00789                 energy += (charge*phi[ivert]*uval);
00790             } /* end for ivert */
00791             /* We only use one simplex of the appropriate color for
00792             * energy calculations, so break here */
00793             break;
00794         } /* endif (color) */
00795     } /* end for isimp */
00796 }
00797
00798     return energy;
00799 }
00800
00801
00802 VPUBLIC double Vfetk_dqmEnergy(Vfetk *thee, int color) {
00803
00804     return AM_evalJ(thee->am);
00805
00806 }
00807
00808 VPUBLIC void Vfetk_setAtomColors(Vfetk *thee) {
00809
00810 #define VMAXLOCALCOLORSDONTREUSETHISVARIABLE 1024
00811     SS *simp;
00812     Vatom *atom;
00813     int i, natoms;
00814
00815     VASSERT(thee != VNULL);
00816
00817     natoms = Valist_getNumberAtoms(thee->pbe->alist);
00818     for (i=0; i<natoms; i++) {
00819         atom = Valist_getAtom(thee->pbe->alist, i);
00820         simp = Vcsm_getSimplex(thee->csm, 0, i);
00821         Vatom_setPartID(atom, SS_chart(simp));
00822     }
00823
00824 }
00825
00826 VPUBLIC unsigned long int Vfetk_memChk(Vfetk *thee) {
00827
00828     int memUse = 0;
00829
00830     if (thee == VNULL) return 0;
00831
00832     memUse = memUse + sizeof(Vfetk);

```

```

00833     memUse = memUse + Vcsm_memChk (thee->csm) ;
00834
00835     return memUse;
00836 }
00837
00838 VPUBLIC Vrc_Codes Vfetk_genCube(
00839     Vfetk *thee,
00840     double center[3],
00841     double length[3],
00842     Vfetk_MeshLoad meshType) {
00843     AM *am = VNULL;
00844     Gem *gm = VNULL;
00845
00846     int skey = 0; /* Simplex format */
00847     char *key = "r"; /* Read */
00848     char *iodev = "BUFF"; /* Buffer */
00849     char *iofmt = "ASC"; /* ASCII */
00850     char *iohost = "localhost"; /* localhost (dummy) */
00851     char *iofile = "0"; /*< socket 0 (dummy) */
00852     Vio *sock = VNULL;
00853     char buf[VMAX_BUFSIZE];
00854     int bufsize = 0;
00855     VV *vx = VNULL;
00856     int i, j;
00857     double x;
00858
00859     VASSERT(thee != VNULL);
00860     am = thee->am;
00861     VASSERT(am != VNULL);
00862     gm = thee->gm;
00863     VASSERT(gm != VNULL);
00864
00865     /* @note This code is based on Gem_makeCube by Mike Holst */
00866     /* Write mesh string to buffer and read back */
00867     switch (meshType) {
00868         case VML_DIRICUBE:
00869             bufsize = strlen(diriCubeString);
00870             VASSERT( bufsize <= VMAX_BUFSIZE );
00871             strncpy(buf, diriCubeString, VMAX_BUFSIZE);
00872             break;
00873         case VML_NEUMCUBE:
00874             bufsize = strlen(neumCubeString);
00875             Vnm_print(2, "Vfetk_genCube: WARNING! USING EXPERIMENTAL NEUMANN BOUNDARY CO-
NDITIONS!\n");
00876             VASSERT( bufsize <= VMAX_BUFSIZE );
00877             strncpy(buf, neumCubeString, VMAX_BUFSIZE);
00878             break;
00879         case VML_EXTERNAL:
00880             Vnm_print(2, "Vfetk_genCube: Got request for external mesh!\n");
00881             Vnm_print(2, "Vfetk_genCube: How did we get here?\n");
00882             return VRC_FAILURE;
00883             break;
00884         default:
00885             Vnm_print(2, "Vfetk_genCube: Unknown mesh type (%d)\n", meshType);
00886             return VRC_FAILURE;
00887     }
00888     VASSERT( VNULL != (sock=Vio_socketOpen(key,iodev,iofmt,iohost,iofile)) );

```

```

00889     Vio_bufTake(sock, buf, bufsize);
00890     AM_read(am, skey, sock);
00891     Vio_connectFree(sock);
00892     Vio_bufGive(sock);
00893     Vio_dtor(&sock);
00894
00895     /* Scale (unit) cube */
00896     for (i=0; i<Gem_numVV(gm); i++) {
00897         vx = Gem_VV(gm, i);
00898         for (j=0; j<3; j++) {
00899             x = VV_coord(vx, j);
00900             x *= length[j];
00901             VV_setCoord(vx, j, x);
00902         }
00903     }
00904
00905     /* Add new center */
00906     for (i=0; i<Gem_numVV(gm); i++) {
00907         vx = Gem_VV(gm, i);
00908         for (j=0; j<3; j++) {
00909             x = VV_coord(vx, j);
00910             x += center[j];
00911             VV_setCoord(vx, j, x);
00912         }
00913     }
00914
00915
00916     return VRC_SUCCESS;
00917 }
00918
00919 VPUBLIC Vrc_Codes Vfetk_loadMesh(
00920     Vfetk *thee,
00921     double center[3],
00922     double length[3],
00923     Vfetk_MeshLoad meshType,
00924     Vio *sock) {
00925
00926     Vrc_Codes vrc;
00927     int skey = 0; /* Simplex format */
00928
00929
00930     switch (meshType) {
00931     case VML_EXTERNAL:
00932         if (sock == VNULL) {
00933             Vnm_print(2, "Vfetk_loadMesh: Got NULL socket!\n");
00934             return VRC_FAILURE;
00935         }
00936         AM_read(thee->am, skey, sock);
00937         Vio_connectFree(sock);
00938         Vio_bufGive(sock);
00939         Vio_dtor(&sock);
00940         break;
00941     case VML_DIRICUBE:
00942         vrc = Vfetk_genCube(thee, center, length, meshType);
00943         if (vrc == VRC_FAILURE) return VRC_FAILURE;
00944         break;
00945     case VML_NEUMCUBE:

```

```

00946     vrc = Vfetk_genCube(thee, center, length, meshType);
00947     if (vrc == VRC_FAILURE) return VRC_FAILURE;
00948     break;
00949 default:
00950     Vnm_print(2, "Vfetk_loadMesh: unrecognized mesh type (%d)!\n",
00951             meshType);
00952     return VRC_FAILURE;
00953 };
00954
00955 /* Setup charge-simplex map */
00956 Vnm_print(0, "Vfetk_ctor2: Constructing Vcsm...\n");
00957 thee->csm = VNULL;
00958 thee->csm = Vcsm_ctor(Vpbe_getValist(thee->pbe), thee->gm);
00959 VASSERT(thee->csm != VNULL);
00960 Vcsm_init(thee->csm);
00961
00962 return VRC_SUCCESS;
00963 }
00964
00965
00966 VPUBLIC void Bmat_printHB( Bmat *thee, char *fname ) {
00967
00968     Mat *Ablock;
00969     MATsym pqsym;
00970     int i, j, jj;
00971     int *IA, *JA;
00972     double *D, *L, *U;
00973     FILE *fp;
00974
00975     char mmtitle[72];
00976     char mmkey[] = {"8charkey"};
00977     int totc = 0, ptrc = 0, indc = 0, valc = 0;
00978     char mxtyp[] = {"RUA"}; /* Real Unsymmetric Assembled */
00979     int nrow = 0, ncol = 0, numZ = 0;
00980     int numZdigits = 0, nrowdigits = 0;
00981     int nptrline = 8, nindline = 8, nvalline = 5;
00982     char ptrfmt[] = {"(8I10)           "}, ptrfmtstr[] = {"%10d"};
00983     char indfmt[] = {"(8I10)           "}, indfmtstr[] = {"%10d"};
00984     char valfmt[] = {"(5E16.8)        "}, valfmtstr[] = {"%16.8E"};
00985
00986     VASSERT( thee->numB == 1 ); /* HARDWIRE FOR NOW */
00987     Ablock = thee->AD[0][0];
00988
00989     VASSERT( Mat_format( Ablock ) == DRC_FORMAT ); /* HARDWIRE FOR NOW */
00990
00991     pqsym = Mat_sym( Ablock );
00992
00993     if ( pqsym == IS_SYM ) {
00994         mxtyp[1] = 'S';
00995     } else if ( pqsym == ISNOT_SYM ) {
00996         mxtyp[1] = 'U';
00997     } else {
00998         VASSERT( 0 ); /* NOT VALID */
00999     }
01000
01001     nrow = Bmat_numRT( thee ); /* Number of rows */
01002     ncol = Bmat_numCT( thee ); /* Number of cols */

```

```

01003     numZ = Bmat_numZT( thee ); /* Number of entries */
01004
01005     nrowdigits = (int) (log( nrow )/log( 10 )) + 1;
01006     numZdigits = (int) (log( numZ )/log( 10 )) + 1;
01007
01008     nptrline = (int) ( 80 / (numZdigits + 1) );
01009     nindline = (int) ( 80 / (nrowdigits + 1) );
01010
01011     sprintf(ptrfmt, "(%dI%d)", nptrline,numZdigits+1);
01012     sprintf(ptrfmtstr,"%%dd",numZdigits+1);
01013     sprintf(indfmt, "(%dI%d)", nindline,nrowdigits+1);
01014     sprintf(indfmtstr,"%%dd",nrowdigits+1);
01015
01016     ptrc = (int) ( ( (ncol + 1) - 1 ) / nptrline ) + 1;
01017     indc = (int) ( (numZ - 1) / nindline ) + 1;
01018     valc = (int) ( (numZ - 1) / nvalline ) + 1;
01019
01020     totc = ptrc + indc + valc;
01021
01022     sprintf( mmtitle, "Sparse '%s' Matrix - Harwell-Boeing Format - '%s'",
01023             thee->name, fname );
01024
01025     /* Step 0: Open the file for writing */
01026
01027     fp = fopen( fname, "w" );
01028     if (fp == VNULL) {
01029         Vnm_print(2,"Bmat_printHB: Ouch couldn't open file <%s>\n",fname);
01030         return;
01031     }
01032
01033     /* Step 1: Print the header information */
01034
01035     fprintf( fp, "%-72s%-8s\n", mmtitle, mmkey );
01036     fprintf( fp, "%14d%14d%14d%14d\n", totc, ptrc, indc, valc, 0 );
01037     fprintf( fp, "%3s%11s%14d%14d%14d\n", mxtyp, " ", nrow, ncol, numZ );
01038     fprintf( fp, "%-16s%-16s%-20s%-20s\n", ptrfmt, indfmt, valfmt, "6E13.5" );
01039
01040     IA = Ablock->IA;
01041     JA = Ablock->JA;
01042     D = Ablock->diag;
01043     L = Ablock->offL;
01044     U = Ablock->offU;
01045
01046     if ( pqsym == IS_SYM ) {
01047
01048         /* Step 2: Print the pointer information */
01049
01050         for (i=0; i<(ncol+1); i++) {
01051             fprintf( fp, ptrfmtstr, Ablock->IA[i] + (i+1) );
01052             if ( (i+1) % nptrline ) == 0 ) {
01053                 fprintf( fp, "\n" );
01054             }
01055         }
01056
01057         if ( (ncol+1) % nptrline ) != 0 ) {
01058             fprintf( fp, "\n" );
01059         }

```

```

01060
01061     /* Step 3: Print the index information */
01062
01063     j = 0;
01064     for (i=0; i<ncol; i++) {
01065         fprintf( fp, indfmtstr, i+1); /* diagonal */
01066         if ( ( (j+1) % nindline ) == 0 ) {
01067             fprintf( fp, "\n" );
01068         }
01069         j++;
01070         for (jj=IA[i]; jj<IA[i+1]; jj++) {
01071             fprintf( fp, indfmtstr, JA[jj] + 1 ); /* lower triangle */
01072             if ( ( (j+1) % nindline ) == 0 ) {
01073                 fprintf( fp, "\n" );
01074             }
01075             j++;
01076         }
01077     }
01078
01079     if ( ( j % nindline ) != 0 ) {
01080         fprintf( fp, "\n" );
01081     }
01082
01083     /* Step 4: Print the value information */
01084
01085     j = 0;
01086     for (i=0; i<ncol; i++) {
01087         fprintf( fp, valfmtstr, D[i] );
01088         if ( ( (j+1) % nvalline ) == 0 ) {
01089             fprintf( fp, "\n" );
01090         }
01091         j++;
01092         for (jj=IA[i]; jj<IA[i+1]; jj++) {
01093             fprintf( fp, valfmtstr, L[jj] );
01094             if ( ( (j+1) % nvalline ) == 0 ) {
01095                 fprintf( fp, "\n" );
01096             }
01097             j++;
01098         }
01099     }
01100
01101     if ( ( j % nvalline ) != 0 ) {
01102         fprintf( fp, "\n" );
01103     }
01104
01105 } else { /* ISNOT_SYM */
01106
01107     VASSERT( 0 ); /* NOT CODED YET */
01108 }
01109
01110 /* Step 5: Close the file */
01111 fclose( fp );
01112 }
01113
01114 VPUBLIC PDE* Vfetk_PDE_ctor(Vfetk *fetk) {
01115     PDE *thee = VNULL;

```

```

01117
01118     thee = Vmem_malloc(fetk->vmem, 1, sizeof(PDE));
01119     VASSERT(thee != VNULL);
01120     VASSERT(Vfetk_PDE_ctor2(thee, fetk));
01121
01122     return thee;
01123 }
01124
01125 VPUBLIC int Vfetk_PDE_ctor2(PDE *thee, Vfetk *fetk) {
01126
01127     int i;
01128
01129     if (thee == VNULL) {
01130         Vnm_print(2, "Vfetk_PDE_ctor2: Got NULL thee!\n");
01131         return 0;
01132     }
01133
01134     /* Store a local copy of the Vfetk class */
01135     var.fetk = fetk;
01136
01137     /* PDE-specific parameters and function pointers */
01138     thee->initAssemble = Vfetk_PDE_initAssemble;
01139     thee->initElement = Vfetk_PDE_initElement;
01140     thee->initFace = Vfetk_PDE_initFace;
01141     thee->initPoint = Vfetk_PDE_initPoint;
01142     thee->Fu = Vfetk_PDE_Fu;
01143     thee->Fu_v = Vfetk_PDE_Fu_v;
01144     thee->DFu_wv = Vfetk_PDE_DFu_wv;
01145     thee->delta = Vfetk_PDE_delta;
01146     thee->u_D = Vfetk_PDE_u_D;
01147     thee->u_T = Vfetk_PDE_u_T;
01148     thee->Ju = Vfetk_PDE_Ju;
01149     thee->vec = 1; /* FIX! */
01150     thee->sym[0][0] = 1;
01151     thee->est[0] = 1.0;
01152     for (i=0; i<VMAX_BDTYPE; i++) thee->bmap[0][i] = i;
01153
01154     /* Manifold-specific function pointers */
01155     thee->bisectEdge = Vfetk_PDE_bisectEdge;
01156     thee->mapBoundary = Vfetk_PDE_mapBoundary;
01157     thee->markSimplex = Vfetk_PDE_markSimplex;
01158     thee->oneChart = Vfetk_PDE_oneChart;
01159
01160     /* Element-specific function pointers */
01161     thee->simplexBasisInit = Vfetk_PDE_simplexBasisInit;
01162     thee->simplexBasisForm = Vfetk_PDE_simplexBasisForm;
01163
01164     return 1;
01165 }
01166
01167 VPUBLIC void Vfetk_PDE_dtor(PDE **thee) {
01168
01169     if ((*thee) != VNULL) {
01170         Vfetk_PDE_dtor2(*thee);
01171     /* TODO: The following line is commented out because at the moment,
01172     there is a seg fault when deallocating at the end of a run. Since
01173     this routine is called only once at the very end, we'll leave it

```

```

01174     commented out. However, this could be a memory leak.
01175     */
01176     /* Vmem_free(var.fetk->vmem, 1, sizeof(PDE), (void **)thee); */
01177     (*thee) = VNULL;
01178 }
01179
01180 }
01181
01182 VPUBLIC void Vfetk_PDE_dtor2(PDE *thee) {
01183     var.fetk = VNULL;
01184 }
01185
01186 VPRIVATE double smooth(int nverts, double dist[VAPBS_NVS], double coeff[VAPBS_NVS
01187 ], int meth) {
01188     int i;
01189     double weight;
01190     double num = 0.0;
01191     double den = 0.0;
01192
01193     for (i=0; i<nverts; i++) {
01194         if (dist[i] < VSMALL) return coeff[i];
01195         weight = 1.0/dist[i];
01196         if (meth == 0) {
01197             num += (weight * coeff[i]);
01198             den += weight;
01199         } else if (meth == 1) {
01200             /* Small coefficients reset the average to 0; we need to break out
01201             * of the loop */
01202             if (coeff[i] < VSMALL) {
01203                 num = 0.0;
01204                 break;
01205             } else {
01206                 num += weight; den += (weight/coeff[i]);
01207             }
01208         } else VASSERT(0);
01209     }
01210
01211     return (num/den);
01212 }
01213 }
01214
01215 VPRIVATE double diel() {
01216
01217     int i, j;
01218     double eps, epsp, epsw, dist[5], coeff[5], srad, swin, *vxp;
01219     Vsurf_Meth srfm;
01220     Vacc *acc;
01221     PBEParm *pbeparm;
01222
01223     epsp = Vpbe_getSoluteDiel(var.fetk->pbe);
01224     epsw = Vpbe_getSolventDiel(var.fetk->pbe);
01225     VASSERT(var.fetk->pbeparm != VNULL);
01226     pbeparm = var.fetk->pbeparm;
01227     srfm = pbeparm->srfm;
01228     srad = pbeparm->srad;
01229     swin = pbeparm->swin;

```

```

01230     acc = var.fetk->pbe->acc;
01231
01232     eps = 0;
01233
01234     if (VABS(epsp - epsw) < VSMALL) return epsp;
01235     switch (srfm) {
01236         case VSM_MOL:
01237             eps = ((epsw-epsp)*Vacc_molAcc(acc, var.xq, srad) + epsp);
01238             break;
01239         case VSM_MOLSMOOTH:
01240             for (i=0; i<var.nverts; i++) {
01241                 dist[i] = 0;
01242                 vx = var.vx[i];
01243                 for (j=0; j<3; j++) {
01244                     dist[i] += VSQR(var.xq[j] - vx[j]);
01245                 }
01246                 dist[i] = VSQRT(dist[i]);
01247                 coeff[i] = (epsw-epsp)*Vacc_molAcc(acc, var.xq, srad) + epsp;
01248             }
01249             eps = smooth(var.nverts, dist, coeff, 1);
01250             break;
01251         case VSM_SPLINE:
01252             eps = ((epsw-epsp)*Vacc_splineAcc(acc, var.xq, swin, 0.0) + epsp);
01253             break;
01254         default:
01255             Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
01256             VASSERT(0);
01257     }
01258
01259     return eps;
01260 }
01261
01262 VPRIvATE double ionacc() {
01263
01264     int i, j;
01265     double dist[5], coeff[5], irad, swin, *vx, accval;
01266     Vsurf_Meth srfm;
01267     Vacc *acc = VNULL;
01268     PBEParm *pbeparm = VNULL;
01269
01270     VASSERT(var.fetk->pbeparm != VNULL);
01271     pbeparm = var.fetk->pbeparm;
01272     srfm = pbeparm->srfm;
01273     irad = Vpbe_getMaxIonRadius(var.fetk->pbe);
01274     swin = pbeparm->swin;
01275     acc = var.fetk->pbe->acc;
01276
01277     if (var.zks2 < VSMALL) return 0.0;
01278     switch (srfm) {
01279         case VSM_MOL:
01280             accval = Vacc_ivdwAcc(acc, var.xq, irad);
01281             break;
01282         case VSM_MOLSMOOTH:
01283             for (i=0; i<var.nverts; i++) {
01284                 dist[i] = 0;
01285                 vx = var.vx[i];
01286                 for (j=0; j<3; j++) {

```

```

01287             dist[i] += VSQR(var.xq[j] - vx[j]);
01288         }
01289         dist[i] = VSQRT(dist[i]);
01290         coeff[i] = Vacc_ivdwAcc(acc, var.xq, irad);
01291     }
01292     accval = smooth(var.nverts, dist, coeff, 1);
01293     break;
01294 case VSM_SPLINE:
01295     accval = Vacc_splineAcc(acc, var.xq, swin, irad);
01296     break;
01297 default:
01298     Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
01299     VASSERT(0);
01300 }
01301
01302 return accval;
01303 }
01304
01305 VPRIIVATE double debye_U(Vpbe *pbe, int d, double x[]) {
01306
01307     double size, *position, charge, xkappa, eps_w, dist, T, pot, val;
01308     int iatom, i;
01309     Valist *alist;
01310     Vatom *atom;
01311
01312     eps_w = Vpbe_getSolventDiel(pbe);
01313     xkappa = (1.0e10)*Vpbe_getXkappa(pbe);
01314     T = Vpbe_getTemperature(pbe);
01315     alist = Vpbe_getValist(pbe);
01316     val = 0;
01317     pot = 0;
01318
01319     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01320         atom = Valist_getAtom(alist, iatom);
01321         position = VatomGetPosition(atom);
01322         charge = Vunit_ec*Vatom_getCharge(atom);
01323         size = (1e-10)*Vatom_getRadius(atom);
01324         dist = 0;
01325         for (i=0; i<d; i++) {
01326             dist += VSQR(position[i] - x[i]);
01327         }
01328         dist = (1.0e-10)*VSQRT(dist);
01329         val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
01330         if (xkappa != 0.0) {
01331             val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
01332         }
01333         pot = pot + val;
01334     }
01335     pot = pot*Vunit_ec/(Vunit_kb*T);
01336
01337     return pot;
01338 }
01339
01340 VPRIIVATE double debye_Udiff(Vpbe *pbe, int d, double x[]) {
01341
01342     double size, *position, charge, eps_p, dist, T, pot, val;
01343     double Ufull;

```

```

01344     int iatom, i;
01345     Valist *alist;
01346     Vatom *atom;
01347
01348     Ufull = debye_U(pbe, d, x);
01349
01350     eps_p = Vpbe_getSoluteDiel(pbe);
01351     T = Vpbe_getTemperature(pbe);
01352     alist = Vpbe_getValist(pbe);
01353     val = 0;
01354     pot = 0;
01355
01356     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01357         atom = Valist_getAtom(alist, iatom);
01358         position = Vatom_getPosition(atom);
01359         charge = Vunit_ec*Vatom_getCharge(atom);
01360         size = (1e-10)*Vatom_getRadius(atom);
01361         dist = 0;
01362         for (i=0; i<d; i++) {
01363             dist += VSQR(position[i] - x[i]);
01364         }
01365         dist = (1.0e-10)*VSQRT(dist);
01366         val = (charge)/(4*VPI*Vunit_eps0*eps_p*dist);
01367         pot = pot + val;
01368     }
01369     pot = pot*Vunit_ec/(Vunit_kb*T);
01370
01371     pot = Ufull - pot;
01372
01373     return pot;
01374 }
01375
01376 VPRIPRIVATE void coulomb(Vpbe *pbe, int d, double pt[], double eps, double *U,
01377     double dU[], double *d2U) {
01378
01379     int iatom, i;
01380     double T, pot, fx, fy, fz, x, y, z, scale;
01381     double *position, charge, dist, dist2, val, vec[3], dUold[3], Uold;
01382     Valist *alist;
01383     Vatom *atom;
01384
01385     /* Initialize variables */
01386     T = Vpbe_getTemperature(pbe);
01387     alist = Vpbe_getValist(pbe);
01388     pot = 0; fx = 0; fy = 0; fz = 0;
01389     x = pt[0]; y = pt[1]; z = pt[2];
01390
01391     /* Calculate */
01392     if (!Vgreen_coulombD(var.green, 1, &x, &y, &z, &pot, &fx, &fy, &fz)) {
01393         Vnm_print(2, "Error calculating Green's function!\n");
01394         VASSERT(0);
01395     }
01396
01397
01398     /* Scale the results */
01399     scale = Vunit_ec/(eps*Vunit_kb*T);
01400     *U = pot*scale;

```

```

01401     *d2U = 0.0;
01402     dU[0] = -fx*scale;
01403     dU[1] = -fy*scale;
01404     dU[2] = -fz*scale;
01405
01406 #if 0
01407     /* Compare with old results */
01408     val = 0.0;
01409     Uold = 0.0; dUold[0] = 0.0; dUold[1] = 0.0; dUold[2] = 0.0;
01410     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01411         atom = Valist_getAtom(alist, iatom);
01412         position = VatomGetPosition(atom);
01413         charge = Vatom_getCharge(atom);
01414         dist2 = 0;
01415         for (i=0; i<d; i++) {
01416             vec[i] = (position[i] - pt[i]);
01417             dist2 += VSQR(vec[i]);
01418         }
01419         dist = VSQRT(dist2);
01420
01421     /* POTENTIAL */
01422     Uold = Uold + charge/dist;
01423
01424     /* GRADIENT */
01425     for (i=0; i<d; i++) dUold[i] = dUold[i] + vec[i]*charge/(dist2*dist);
01426
01427 }
01428 Uold = Uold*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*Vunit_kb*T);
01429 for (i=0; i<d; i++) {
01430     dUold[i] = dUold[i]*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*
01431     Vunit_kb*T);
01432 }
01433 printf("Unew - Uold = %g - %g = %g\n", *U, Uold, (*U - Uold));
01434 printf("||dUnew - dUold||^2 = %g\n", (VSQR(dU[0] - dUold[0])
01435             + VSQR(dU[1] - dUold[1]) + VSQR(dU[2] - dUold[2])));
01436 printf("dUnew[0] = %g, dUold[0] = %g\n", dU[0], dUold[0]);
01437 printf("dUnew[1] = %g, dUold[1] = %g\n", dU[1], dUold[1]);
01438 printf("dUnew[2] = %g, dUold[2] = %g\n", dU[2], dUold[2]);
01439
01440 #endif
01441
01442 }
01443
01444 VPUBLIC void Vfetk_PDE_initAssemble(PDE *thee, int ip[], double rp[]) {
01445
01446 #if 1
01447     /* Re-initialize the Green's function oracle in case the atom list has
01448      * changed */
01449     if (var.initGreen) {
01450         Vgreen_dtor(&(var.green));
01451         var.initGreen = 0;
01452     }
01453     var.green = Vgreen_ctor(var.fetk->pbe->alist);
01454     var.initGreen = 1;
01455 #else
01456     if (!var.initGreen) {

```

```

01457     var.green = Vgreen_ctor(var.fetk->pbe->alist);
01458     var.initGreen = 1;
01459 }
01460 #endif
01461
01462 }
01463
01464 VPUBLIC void Vfetk_PDE_initElement(PDE *thee, int elementType, int chart,
01465   double tvx[][3], void *data) {
01466
01467   int i, j;
01468   double epsp, epsw;
01469
01470   /* We assume that the simplex has been passed in as the void *data */
01471   /* argument. Store it */
01472   VASSERT(data != NULL);
01473   var.simp = (SS *)data;
01474
01475   /* save the element type */
01476   var.sType = elementType;
01477
01478   /* Grab the vertices from this simplex */
01479   var.nverts = thee->dim+1;
01480   for (i=0; i<thee->dim+1; i++) var.verts[i] = SS_vertex(var.simp, i);
01481
01482   /* Vertex locations of this simplex */
01483   for (i=0; i<thee->dim+1; i++) {
01484     for (j=0; j<thee->dim; j++) {
01485       var.vx[i][j] = tvx[i][j];
01486     }
01487   }
01488
01489   /* Set the dielectric constant for this element for use in the jump term */
01490   /* of the residual-based error estimator. The value is set to the average
01491   /* * value of the vertices */
01492   var.jumpDiel = 0; /* NOT IMPLEMENTED YET! */
01493 }
01494
01495 VPUBLIC void Vfetk_PDE_initFace(PDE *thee, int faceType, int chart,
01496   double tnvec[]) {
01497
01498   int i;
01499
01500   /* unit normal vector of this face */
01501   for (i=0; i<thee->dim; i++) var.nvec[i] = tnvec[i];
01502
01503   /* save the face type */
01504   var.fType = faceType;
01505 }
01506
01507 VPUBLIC void Vfetk_PDE_initPoint(PDE *thee, int pointType, int chart,
01508   double txq[], double tU[], double tdU[][3]) {
01509
01510   int i, j, ichop;
01511   double u2, coef2, eps_p;
01512   Vhal_PBEType pdetype;
01513   Vpbe *pbe = VNULL;

```

```

01514
01515     eps_p = Vpbe_getSoluteDiel (var.fetk->pbe);
01516     pdetype = var.fetk->type;
01517     pbe = var.fetk->pbe;
01518
01519     /* the point, the solution value and gradient, and the Coulomb value and *
01520      * gradient at the point */
01521     if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01522         coulomb(pbe, thee->dim, txq, eps_p, &(var.W), var.dW, &(var.d2W));
01523     }
01524     for (i=0; i<thee->vec; i++) {
01525         var.U[i] = tU[i];
01526         for (j=0; j<thee->dim; j++) {
01527             var.xq[j] = txq[j];
01528             var.dU[i][j] = tdU[i][j];
01529         }
01530     }
01531
01532     /* interior form case */
01533     if (pointType == 0) {
01534
01535         /* Get the dielectric values */
01536         var.diel = diel();
01537         var.ionacc = ionacc();
01538         var.A = var.diel;
01539         var.F = (var.diel - eps_p);
01540
01541         switch (pdetype) {
01542
01543             case PBE_LPBE:
01544                 var.DB = var.ionacc*var.zkappa2*var.ionstr;
01545                 var.B = var.DB*var.U[0];
01546                 break;
01547
01548             case PBE_NPBE:
01549
01550                 var.B = 0;
01551                 var.DB = 0;
01552                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01553                     for (i=0; i<var.nion; i++) {
01554                         u2 = -1.0 * var.U[0] * var.ionQ[i];
01555
01556                         /* NONLINEAR TERM */
01557                         coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01558                         var.B += (coef2 * Vcap_exp(u2, &ichop));
01559                         /* LINEARIZED TERM */
01560                         coef2 = -1.0 * var.ionQ[i] * coef2;
01561                         var.DB += (coef2 * Vcap_exp(u2, &ichop));
01562                     }
01563                 }
01564                 break;
01565
01566             case PBE_LRPBE:
01567                 var.DB = var.ionacc*var.zkappa2*var.ionstr;
01568                 var.B = var.DB*(var.U[0]+var.W);
01569                 break;
01570

```

```

01571     case PBE_NRPBE:
01572
01573         var.B = 0;
01574         var.DB = 0;
01575         if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01576             for (i=0; i<var.nion; i++) {
01577                 u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
01578
01579                 /* NONLINEAR TERM */
01580                 coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01581                 var.B += (coef2 * Vcap_exp(u2, &ichop));
01582
01583                 /* LINEARIZED TERM */
01584                 coef2 = -1.0 * var.ionQ[i] * coef2;
01585                 var.DB += (coef2 * Vcap_exp(u2, &ichop));
01586             }
01587         }
01588         break;
01589
01590     case PBE_SMPBE: /* SMPBE Temp */
01591
01592         var.B = 0;
01593         var.DB = 0;
01594         if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01595             for (i=0; i<var.nion; i++) {
01596                 u2 = -1.0 * var.U[0] * var.ionQ[i];
01597
01598                 /* NONLINEAR TERM */
01599                 coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01600                 var.B += (coef2 * Vcap_exp(u2, &ichop));
01601                 /* LINEARIZED TERM */
01602                 coef2 = -1.0 * var.ionQ[i] * coef2;
01603                 var.DB += (coef2 * Vcap_exp(u2, &ichop));
01604             }
01605         }
01606         break;
01607     default:
01608         Vnm_print(2, "Vfetk_PDE_initPoint: Unknown PBE type (%d)!\n",
01609                  pdetype);
01610         VASSERT(0);
01611         break;
01612     }
01613
01614
01615     /* boundary form case */
01616     } else {
01617 #ifdef DONEUMANN
01618     ;
01619 #else
01620     Vnm_print(2, "Vfetk: Whoa! I just got a boundary point to evaluate (%d)
01621 !\n", pointType);
01622     Vnm_print(2, "Vfetk: Did you do that on purpose?\n");
01623 #endif
01624     }
01625 #if 0 /* THIS IS VERY NOISY! */
01626     Vfetk_dumpLocalVar();

```

```

01627 #endif
01628
01629 }
01630
01631 VPUBLIC void Vfetk_PDE_Fu(PDE *thee, int key, double F[]) {
01632     //Vnm_print(2, "Vfetk_PDE_Fu: Setting error to zero!\n");
01633     F[0] = 0.;
01634
01635 }
01636
01637 }
01638
01639 VPUBLIC double Vfetk_PDE_Fu_v(
01640     PDE *thee,
01641     int key,
01642     double V[],
01643     double dV[] [VAPBS_DIM]
01644 )
01645
01646     Vhal_PBEType type;
01647     int i;
01648     double value = 0.;
01649
01650     type = var.fetk->type;
01651
01652     /* interior form case */
01653     if (key == 0) {
01654
01655         for (i=0; i<thee->dim; i++) value += (var.A * var.dU[0][i] * dV[0][i]);
01656
01657         value += var.B * V[0];
01658
01659         if ((type == PBE_LRPBE) || (type == PBE_NRPBE)) {
01660             for (i=0; i<thee->dim; i++) {
01661                 if (var.F > VSMALL) value += (var.F * var.dW[i] * dV[0][i]);
01662             }
01663
01664         /* boundary form case */
01665         } else {
01666 #ifdef DONEUMANN
01667             value = 0.0;
01668 #else
01669             Vnm_print(2, "Vfetk: Whoa! I was just asked to evaluate a boundary weak
form for point type %d!\n", key);
01670 #endif
01671     }
01672
01673     var.Fu_v = value;
01674     return value;
01675 }
01676
01677 VPUBLIC double Vfetk_PDE_DFu_wv(
01678     PDE *thee,
01679     int key,
01680     double W[],
01681     double dW[] [VAPBS_DIM],

```

```

01682     double V[],
01683     double dV[][][3]
01684 ) {
01685
01686     Vhal_PBEType type;
01687     int i;
01688     double value = 0.;
01689
01690     type = var.fetk->type;
01691
01692     /* Interior form */
01693     if (key == 0) {
01694         value += var.DB * W[0] * V[0];
01695         for (i=0; i<thee->dim; i++) value += ( var.A * dW[0][i] * dV[0][i] );
01696
01697     /* boundary form case */
01698     } else {
01699 #ifdef DONEUMANN
01700         value = 0.0;
01701     #else
01702         Vnm_print(2, "Vfetk: Whoa! I was just asked to evaluate a boundary weak
01703         form for point type %d!\n", key);
01704     #endif
01705 }
01706
01707     var.DFu_wv = value;
01708     return value;
01709 }
01710
01711 #define VRINGMAX 1000
01712
01713 #define VATOMMAX 1000000
01714 VPUBLIC void Vfetk_PDE_delta(PDE *thee, int type, int chart, double txq[],
01715     void *user, double F[]) {
01716
01717     int iatom, jatom, natoms, atomIndex, atomList[VATOMMAX], nAtomList;
01718     int gotAtom, numSring, isimp, invert, sid;
01719     double *position, charge, phi[VAPBS_NVS], phix[VAPBS_NVS][3], value;
01720     Vatom *atom;
01721     Vhal_PBEType pdetype;
01722     SS *sring[VRINGMAX];
01723     VV *vertex = (VV *)user;
01724
01725     pdetype = var.fetk->type;
01726
01727     F[0] = 0.0;
01728
01729     if ((pdetype == PBE_LPBE) || (pdetype == PBE_NPBE) || (pdetype == PBE_SMPBE)
01730      /* SMPBE Added */ {
01731         VASSERT( vertex != VNNULL );
01732         numSring = 0;
01733         sring[numSring] = VV_firstSS(vertex);
01734         while (sring[numSring] != VNNULL) {
01735             numSring++;
01736             sring[numSring] = SS_link(sring[numSring-1], vertex);
01737         }
01738     }

```

```

01739         VASSERT( numSring > 0 );
01740         VASSERT( numSring <= VRINGMAX );
01741
01742             /* Move around the simplex ring and determine the charge locations */
01743             F[0] = 0.;
01744             charge = 0.;
01745             nAtomList = 0;
01746             for (isimp=0; isimp<numSring; isimp++) {
01747                 sid = SS_id(sring[isimp]);
01748                 natoms = Vcsm_getNumberAtoms(Vfetk_getVcsm(var.fetk), sid);
01749                 for (iatom=0; iatom<natoms; iatom++) {
01750                     /* Get the delta function information */
01751                     atomIndex = Vcsm_getAtomIndex(Vfetk_getVcsm(var.fetk),
01752                         iatom, sid);
01753                     gotAtom = 0;
01754                     for (jatom=0; jatom<nAtomList; jatom++) {
01755                         if (atomList[jatom] == atomIndex) {
01756                             gotAtom = 1;
01757                             break;
01758                         }
01759                     }
01760                     if (!gotAtom) {
01761                         VASSERT(nAtomList < VATOMMAX);
01762                         atomList[nAtomList] = atomIndex;
01763                         nAtomList++;
01764
01765                         atom = Vcsm_getAtom(Vfetk_getVcsm(var.fetk), iatom, sid);
01766                         charge = Vatom_getCharge(atom);
01767                         position = Vatom_getPosition(atom);
01768
01769                         /* Get the test function value at the delta function I
01770                          * used to do a VASSERT to make sure the point was in the
01771                          * simplex (i.e., make sure round-off error isn't an
01772                          * issue), but round off errors became an issue */
01773                         if (!Gem_pointInSimplexVal(Vfetk_getGem(var.fetk),
01774                             sring[isimp], position, phi, phix)) {
01775                             if (!Gem_pointInSimplex(Vfetk_getGem(var.fetk),
01776                                 sring[isimp], position)) {
01777                                 Vnm_print(2, "delta: Both Gem_pointInSimplexVal \
01778 and Gem_pointInSimplex detected misplaced point charge!\n");
01779                                 Vnm_print(2, "delta: I think you have problems: \
01780 phi = \"%");
01781                             for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.fetk))
01782                             ; ivert++) Vnm_print(2, "%e ", phi[ivert]);
01783
01784                             Vnm_print(2, "}\n");
01785                         }
01786                         value = 0;
01787                         for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.fetk)); ivert+
01788                         ) {
01789                             if (VV_id(SS_vertex(sring[isimp], ivert)) == VV_id(vertex
01790 )) value += phi[ivert];
01791                         }
01792
01793                         F[0] += (value * Vpbe_getZmagic(var.fetk->pbe) * charge);
01794 } /* if !gotAtom */

```

```

01792             } /* for iatom */
01793         } /* for isimp */
01794
01795     } else if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01796         F[0] = 0.0;
01797     } else { VASSERT(0); }
01798
01799     var.delta = F[0];
01800
01801 }
01802
01803 VPUBLIC void Vfetk_PDE_u_D(PDE *thee, int type, int chart, double txq[],
01804     double F[]) {
01805
01806     if ((var.fetk->type == PBE_LPBE) || (var.fetk->type == PBE_NPBE) || (var.
01807         fetk->type == PBE_SMPBE) /* SMPBE Added */)
01808         F[0] = debye_U(var.fetk->pbe, thee->dim, txq);
01809     } else if ((var.fetk->type == PBE_LRPBE) || (var.fetk->type == PBE_NRPBE)) {
01810         F[0] = debye_Udiff(var.fetk->pbe, thee->dim, txq);
01811     } else VASSERT(0);
01812
01813     var.u_D = F[0];
01814 }
01815
01816 VPUBLIC void Vfetk_PDE_u_T(PDE *thee, int type, int chart, double txq[],
01817     double F[]) {
01818
01819     F[0] = 0.0;
01820     var.u_T = F[0];
01821
01822 }
01823
01824
01825 VPUBLIC void Vfetk_PDE_bisectEdge(int dim, int dimII, int edgeType,
01826     int chart[], double vx[][3]) {
01827
01828     int i;
01829
01830     for (i=0; i<dimII; i++) vx[2][i] = .5 * (vx[0][i] + vx[1][i]);
01831     chart[2] = chart[0];
01832
01833 }
01834
01835 VPUBLIC void Vfetk_PDE_mapBoundary(int dim, int dimII, int vertexType,
01836     int chart, double vx[3]) {
01837
01838 }
01839
01840 VPUBLIC int Vfetk_PDE_markSimplex(int dim, int dimII, int simplexType,
01841     int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][3]
01842     ,
01843     void *simplex) {
01844
01845     double targetRes, edgeLength, srad, swin, myAcc, refAcc;
01846     int i, natoms;
01847     Vsurf_Meth srfm;

```

```

01847     Vhal_PBEType type;
01848     FEMparm *feparm = VNULL;
01849     PBEparm *pbeparm = VNULL;
01850     Vpbe *pbe = VNULL;
01851     Vacc *acc = VNULL;
01852     Vcsm *csm = VNULL;
01853     SS *simp = VNULL;
01854
01855     VASSERT(var.fetk->feparm != VNULL);
01856     feparm = var.fetk->feparm;
01857     VASSERT(var.fetk->pbeparm != VNULL);;
01858     pbeparm = var.fetk->pbeparm;
01859     pbe = var.fetk->pbe;
01860     csm = Vfetk_getVcsm(var.fetk);
01861     acc = pbe->acc;
01862     targetRes = feparm->targetRes;
01863     srfm = pbeparm->srfm;
01864     srad = pbeparm->srad;
01865     swin = pbeparm->swin;
01866     simp = (SS *)simplex;
01867     type = var.fetk->type;
01868
01869     /* Check to see if this simplex is smaller than the target size */
01870     /* NAB WARNING: I am providing face=-1 here to conform to the new MC API; ho
    wever, I'm not sure if this is the correct behavior. */
01871     Gem_longestEdge(var.fetk->gm, simp, -1, &edgeLength);
01872     if (edgeLength < targetRes) return 0;
01873
01874     /* For non-regularized PBE, check charge-simplex map */
01875     if ((type == PBE_LPBE) || (type == PBE_NPBE) || (type == PBE_SMPBE) /* SMPBE
    Added */) {
01876         natoms = Vcsm_getNumberAtoms(csm, SS_id(simp));
01877         if (natoms > 0) {
01878             return 1;
01879         }
01880     }
01881
01882     /* We would like to resolve the mesh between the van der Waals surface the
    * max distance from this surface where there could be coefficient
    * changes */
01883     switch(srfm) {
01884         case VSM_MOL:
01885             refAcc = Vacc_molAcc(acc, vx[0], srad);
01886             for (i=1; i<(dim+1); i++) {
01887                 myAcc = Vacc_molAcc(acc, vx[i], srad);
01888                 if (myAcc != refAcc) {
01889                     return 1;
01890                 }
01891             }
01892             break;
01893         case VSM_MOLSMOOTH:
01894             refAcc = Vacc_molAcc(acc, vx[0], srad);
01895             for (i=1; i<(dim+1); i++) {
01896                 myAcc = Vacc_molAcc(acc, vx[i], srad);
01897                 if (myAcc != refAcc) {
01898                     return 1;
01899                 }
01900             }
01901     }

```

```

01902         }
01903         break;
01904     case VSM_SPLINE:
01905         refAcc = Vacc_splineAcc(acc, vx[0], swin, 0.0);
01906         for (i=1; i<(dim+1); i++) {
01907             myAcc = Vacc_splineAcc(acc, vx[i], swin, 0.0);
01908             if (myAcc != refAcc) {
01909                 return 1;
01910             }
01911         }
01912         break;
01913     default:
01914         VASSERT(0);
01915         break;
01916     }
01917
01918     return 0;
01919 }
01920
01921 VPUBLIC void Vfetk_PDE_oneChart(int dim, int dimII, int objType, int chart[],
01922     double vx[][][3], int dimV) {
01923
01924 }
01925
01926 VPUBLIC double Vfetk_PDE_Ju(PDE *thee, int key) {
01927
01928     int i, ichop;
01929     double dielE, qmE, coef2, u2;
01930     double value = 0.;
01931     Vhal_PBEType type;
01932
01933     type = var.fetk->type;
01934
01935     /* interior form case */
01936     if (key == 0) {
01937         dielE = 0;
01938         for (i=0; i<3; i++) dielE += VSQR(var.dU[0][i]);
01939         dielE = dielE*var.diel;
01940
01941         switch (type) {
01942             case PBE_LPBE:
01943                 if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
01944                     qmE = var.ionacc*var.zkappa2*VSQR(var.U[0]);
01945                 } else qmE = 0;
01946                 break;
01947             case PBE_NPBE:
01948                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01949                     qmE = 0.;
01950                     for (i=0; i<var.nion; i++) {
01951                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.
01952                         ionQ[i];
01953                         u2 = -1.0 * (var.U[0]) * var.ionQ[i];
01954                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
01955                     }
01956                 } else qmE = 0;
01957                 break;
01958             case PBE_LRPBE:

```

```

01958         if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
01959             qmE = var.ionacc*var.zkappa2*VSQR((var.U[0] + var.W));
01960         } else qmE = 0;
01961         break;
01962     case PBE_NRPBE:
01963         if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01964             qmE = 0.;
01965             for (i=0; i<var.nion; i++) {
01966                 coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.
01967                     ionQ[i];
01968                 u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
01969                 qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
01970             } else qmE = 0;
01971             break;
01972     case PBE_SMPBE: /* SMPBE Temp */
01973         if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01974             qmE = 0.;
01975             for (i=0; i<var.nion; i++) {
01976                 coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.
01977                     ionQ[i];
01978                 u2 = -1.0 * (var.U[0]) * var.ionQ[i];
01979                 qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
01980             } else qmE = 0;
01981             break;
01982     default:
01983         Vnm_print(2, "Vfetk_PDE_Ju: Invalid PBE type (%d)!\n", type);
01984         VASSERT(0);
01985         break;
01986     }
01987     value = 0.5*(dielE + qmE)/Vpbe_getZmagic(var.fetk->pbe);
01988
01989 /* boundary form case */
01990 } else if (key == 1) {
01991     value = 0.0;
01992
01993 /* how did we get here? */
01994 } else VASSERT(0);
01995
01996 return value;
01997
01998
01999 }
02000
02001 VPUBLIC void Vfetk_externalUpdateFunction(SS **simps, int num) {
02002
02003     Vcsm *csm = VNULL;
02004     int rc;
02005
02006     VASSERT(var.fetk != VNULL);
02007     csm = Vfetk_getVcsm(var.fetk);
02008     VASSERT(csm != VNULL);
02009
02010     rc = Vcsm_update(csm, simps, num);
02011
02012     if (!rc) {

```

```

02013     Vnm_print(2, "Error while updating charge-simplex map!\n");
02014     VASSERT(0);
02015 }
02016 }
02017
02018 VPRIIVATE void polyEval(int numP, double p[], double c[][VMAXP], double xv[]) {
02019     int i;
02020     double x, y, z;
02021
02022     x = xv[0];
02023     y = xv[1];
02024     z = xv[2];
02025     for (i=0; i<numP; i++) {
02026         p[i] = c[i][0]
02027             + c[i][1] * x
02028             + c[i][2] * y
02029             + c[i][3] * z
02030             + c[i][4] * x*x
02031             + c[i][5] * y*y
02032             + c[i][6] * z*z
02033             + c[i][7] * x*y
02034             + c[i][8] * x*z
02035             + c[i][9] * y*z
02036             + c[i][10] * x*x*x
02037             + c[i][11] * y*y*y
02038             + c[i][12] * z*z*z
02039             + c[i][13] * x*x*y
02040             + c[i][14] * x*x*z
02041             + c[i][15] * x*y*y
02042             + c[i][16] * y*y*z
02043             + c[i][17] * x*z*z
02044             + c[i][18] * y*z*z;
02045     }
02046 }
02047
02048 VPRIIVATE void setCoef(int numP, double c[][VMAXP], double cx[][VMAXP],
02049     double cy[][VMAXP], double cz[][VMAXP], int ic[][VMAXP], int icx[][VMAXP],
02050     int icy[], int icz[]) {
02051
02052     int i, j;
02053     for (i=0; i<numP; i++) {
02054         for (j=0; j<VMAXP; j++) {
02055             c[i][j] = 0.5 * (double)ic[i][j];
02056             cx[i][j] = 0.5 * (double)icx[i][j];
02057             cy[i][j] = 0.5 * (double)icy[i][j];
02058             cz[i][j] = 0.5 * (double)icz[i][j];
02059         }
02060     }
02061 }
02062
02063 VPUBLIC int Vfetk_PDE_simplexBasisInit(int key, int dim, int comp, int *ndof,
02064     int dof[]) {
02065
02066     int qorder, bump, dimIS[VAPBS_NVS];
02067
02068     /* necessary quadrature order to return at the end */
02069     qorder = P_DEG;

```

```

02070
02071     /* deal with bump function requests */
02072     if ((key == 0) || (key == 1)) {
02073         bump = 0;
02074     } else if ((key == 2) || (key == 3)) {
02075         bump = 1;
02076     } else { VASSERT(0); }
02077
02078     /* for now use same element for all components, both trial and test */
02079     if (dim==2) {
02080         /* 2D simplex dimensions */
02081         dimIS[0] = 3; /* number of vertices */ */
02082         dimIS[1] = 3; /* number of edges */ */
02083         dimIS[2] = 0; /* number of faces (3D only) */ */
02084         dimIS[3] = 1; /* number of simplices (always=1) */ */
02085         if (bump==0) {
02086             if (P_DEG==1) {
02087                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02088             } else if (P_DEG==2) {
02089                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02090             } else if (P_DEG==3) {
02091                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02092             } else Vnm_print(2, "...bad order..");
02093             } else if (bump==1) {
02094                 if (P_DEG==1) {
02095                     init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02096                 } else Vnm_print(2, "...bad order..");
02097             } else Vnm_print(2, "...bad bump..");
02098         } else if (dim==3) {
02099             /* 3D simplex dimensions */
02100             dimIS[0] = 4; /* number of vertices */ */
02101             dimIS[1] = 6; /* number of edges */ */
02102             dimIS[2] = 4; /* number of faces (3D only) */ */
02103             dimIS[3] = 1; /* number of simplices (always=1) */ */
02104             if (bump==0) {
02105                 if (P_DEG==1) {
02106                     init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02107                 } else if (P_DEG==2) {
02108                     init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02109                 } else if (P_DEG==3) {
02110                     init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02111                 } else Vnm_print(2, "...bad order..");
02112             } else if (bump==1) {
02113                 if (P_DEG==1) {
02114                     init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02115                 } else Vnm_print(2, "...bad order..");
02116             } else Vnm_print(2, "...bad bump..");
02117         } else Vnm_print(2, "...bad dimension..");
02118
02119         /* save number of DF */
02120         numP = *ndof;
02121
02122         /* return the required quadrature order */
02123         return qorder;
02124     }
02125
02126 VPUBLIC void Vfetk_PDE_simplexBasisForm(int key, int dim, int comp, int pdkey,

```

```

02127     double xq[], double basis[]) {
02128
02129     if (pdkey == 0) {
02130         polyEval(numP, basis, c, xq);
02131     } else if (pdkey == 1) {
02132         polyEval(numP, basis, cx, xq);
02133     } else if (pdkey == 2) {
02134         polyEval(numP, basis, cy, xq);
02135     } else if (pdkey == 3) {
02136         polyEval(numP, basis, cz, xq);
02137     } else { VASSERT(0); }
02138 }
02139
02140 VPRIIVATE void init_2DP1(int dimIS[], int *ndof, int dof[], double c[] [VMAXP],
02141     double cx[] [VMAXP], double cy[] [VMAXP], double cz[] [VMAXP]) {
02142
02143     int i;
02144
02145     /* dof number and locations */
02146     dof[0] = 1;
02147     dof[1] = 0;
02148     dof[2] = 0;
02149     dof[3] = 0;
02150     *ndof = 0;
02151     for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
02152     VASSERT( *ndof == dim_2DP1 );
02153     VASSERT( *ndof <= VMAXP );
02154
02155     /* coefficients of the polynomials */
02156     setCoef( *ndof, c, cx, cy, cz, lgr_2DP1, lgr_2DP1x, lgr_2DP1y, lgr_2DP1z );
02157 }
02158
02159 VPRIIVATE void init_3DP1(int dimIS[], int *ndof, int dof[], double c[] [VMAXP],
02160     double cx[] [VMAXP], double cy[] [VMAXP], double cz[] [VMAXP]) {
02161
02162     int i;
02163
02164     /* dof number and locations */
02165     dof[0] = 1;
02166     dof[1] = 0;
02167     dof[2] = 0;
02168     dof[3] = 0;
02169     *ndof = 0;
02170     for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
02171     VASSERT( *ndof == dim_3DP1 );
02172     VASSERT( *ndof <= VMAXP );
02173
02174     /* coefficients of the polynomials */
02175     setCoef( *ndof, c, cx, cy, cz, lgr_3DP1, lgr_3DP1x, lgr_3DP1y, lgr_3DP1z );
02176 }
02177
02178 VPUBLIC void Vfetk_dumpLocalVar() {
02179
02180     int i;
02181
02182     Vnm_print(1, "DEBUG: nvec = (%g, %g, %g)\n", var.nvec[0], var.nvec[1],
02183             var.nvec[2]);

```

```

02184     Vnm_print(1, "DEBUG: nverts = %d\n", var.nverts);
02185     for (i=0; i<var.nverts; i++) {
02186         Vnm_print(1, "DEBUG: verts[%d] ID = %d\n", i, VV_id(var.verts[i]));
02187         Vnm_print(1, "DEBUG: vx[%d] = (%g, %g, %g)\n", i, var.vx[i][0],
02188             var.vx[i][1], var.vx[i][2]);
02189     }
02190     Vnm_print(1, "DEBUG: simp ID = %d\n", SS_id(var.simp));
02191     Vnm_print(1, "DEBUG: sType = %d\n", var.sType);
02192     Vnm_print(1, "DEBUG: fType = %d\n", var.fType);
02193     Vnm_print(1, "DEBUG: xq = (%g, %g, %g)\n", var.xq[0], var.xq[1], var.xq[2]);
02194     Vnm_print(1, "DEBUG: U[0] = %g\n", var.U[0]);
02195     Vnm_print(1, "DEBUG: dU[0] = (%g, %g, %g)\n", var.dU[0][0], var.dU[0][1],
02196             var.dU[0][2]);
02197     Vnm_print(1, "DEBUG: W = %g\n", var.W);
02198     Vnm_print(1, "DEBUG: d2W = %g\n", var.d2W);
02199     Vnm_print(1, "DEBUG: dW = (%g, %g, %g)\n", var.dW[0], var.dW[1], var.dW[2]);
02200     Vnm_print(1, "DEBUG: diel = %g\n", var.diel);
02201     Vnm_print(1, "DEBUG: ionacc = %g\n", var.ionacc);
02202     Vnm_print(1, "DEBUG: A = %g\n", var.A);
02203     Vnm_print(1, "DEBUG: F = %g\n", var.F);
02204     Vnm_print(1, "DEBUG: B = %g\n", var.B);
02205     Vnm_print(1, "DEBUG: DB = %g\n", var.DB);
02206     Vnm_print(1, "DEBUG: nion = %d\n", var.nion);
02207     for (i=0; i<var.nion; i++) {
02208         Vnm_print(1, "DEBUG: ionConc[%d] = %g\n", i, var.ionConc[i]);
02209         Vnm_print(1, "DEBUG: ionQ[%d] = %g\n", i, var.ionQ[i]);
02210         Vnm_print(1, "DEBUG: ionRadii[%d] = %g\n", i, var.ionRadii[i]);
02211     }
02212     Vnm_print(1, "DEBUG: zkappa2 = %g\n", var.zkappa2);
02213     Vnm_print(1, "DEBUG: zks2 = %g\n", var.zks2);
02214     Vnm_print(1, "DEBUG: Fu_v = %g\n", var.Fu_v);
02215     Vnm_print(1, "DEBUG: DFu_wv = %g\n", var.DFu_wv);
02216     Vnm_print(1, "DEBUG: delta = %g\n", var.delta);
02217     Vnm_print(1, "DEBUG: u_D = %g\n", var.u_D);
02218     Vnm_print(1, "DEBUG: u_T = %g\n", var.u_T);
02219
02220 };
02221
02222 VPUBLIC int Vfetk_fillArray(Vfetk *thee, Bvec *vec, Vdata_Type type) {
02223
02224     int i, j, ichop;
02225     double coord[3], chi, q, conc, val;
02226     VV *vert;
02227     Bvec *u, *u_d;
02228     AM *am;
02229     Gem *gm;
02230     PBEParm *pbeparm;
02231     Vacc *acc;
02232     Vpbe *pbe;
02233
02234     gm = thee->gm;
02235     am = thee->am;
02236     pbe = thee->pbe;
02237     pbeparm = thee->pbeparm;
02238     acc = pbe->acc;
02239
02240     /* Make sure vec has enough rows to accomodate the vertex data */

```

```

02241     if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02242         Vnm_print(2, "Vfetk_fillArray: insufficient space in Bvec!\n");
02243         Vnm_print(2, "Vfetk_fillArray: Have %d, need %d!\n", Bvec_numRB(vec, 0),
02244                     Gem_numVV(gm));
02245         return 0;
02246     }
02247
02248     switch (type) {
02249
02250         case VDT_CHARGE:
02251             Vnm_print(2, "Vfetk_fillArray: can't write out charge distribution!\
n");
02252             return 0;
02253             break;
02254
02255         case VDT_POT:
02256             u = am->u;
02257             u_d = am->ud;
02258             /* Copy in solution */
02259             Bvec_copy(vec, u);
02260             /* Add dirichlet condition */
02261             Bvec_axpy(vec, u_d, 1.0);
02262             break;
02263
02264         case VDT_SMOL:
02265             for (i=0; i<Gem_numVV(gm); i++) {
02266                 vert = Gem_VV(gm, i);
02267                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02268                 chi = Vacc_molAcc(acc, coord, pbe->solventRadius);
02269                 Bvec_set(vec, i, chi);
02270             }
02271             break;
02272
02273         case VDT_SSPL:
02274             for (i=0; i<Gem_numVV(gm); i++) {
02275                 vert = Gem_VV(gm, i);
02276                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02277                 chi = Vacc_splineAcc(acc, coord, pbeparm->swin, 0.0);
02278                 Bvec_set(vec, i, chi);
02279             }
02280             break;
02281
02282         case VDT_VDW:
02283             for (i=0; i<Gem_numVV(gm); i++) {
02284                 vert = Gem_VV(gm, i);
02285                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02286                 chi = Vacc_vdwAcc(acc, coord);
02287                 Bvec_set(vec, i, chi);
02288             }
02289             break;
02290
02291         case VDT_IVDW:
02292             for (i=0; i<Gem_numVV(gm); i++) {
02293                 vert = Gem_VV(gm, i);
02294                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02295                 chi = Vacc_ivdwAcc(acc, coord, pbe->maxIonRadius);

```

```

02296             Bvec_set(vec, i, chi);
02297         }
02298         break;
02299
02300     case VDT_LAP:
02301         Vnm_print(2, "Vfetk_fillArray: can't write out Laplacian!\n");
02302         return 0;
02303         break;
02304
02305     case VDT_EDENS:
02306         Vnm_print(2, "Vfetk_fillArray: can't write out energy density!\n");
02307         return 0;
02308         break;
02309
02310     case VDT_NDENS:
02311         u = am->u;
02312         u_d = am->ud;
02313         /* Copy in solution */
02314         Bvec_copy(vec, u);
02315         /* Add dirichlet condition */
02316         Bvec_axpy(vec, u_d, 1.0);
02317         /* Load up ions */
02318         ichop = 0;
02319         for (i=0; i<Gem_numVV(gm); i++) {
02320             val = 0;
02321             for (j=0; j<pbe->numIon; j++) {
02322                 q = pbe->ionQ[j];
02323                 conc = pbe->ionConc[j];
02324                 if (thee->type == PBE_NPBE || thee->type == PBE_SMPBE /* SMPB
E Added */) {
02325                     val += (conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));
02326                 } else if (thee->type == PBE_LPBE) {
02327                     val += (conc * (1 - q*Bvec_val(vec, i)));
02328                 }
02329             }
02330             Bvec_set(vec, i, val);
02331         }
02332         break;
02333
02334     case VDT_QDENS:
02335         u = am->u;
02336         u_d = am->ud;
02337         /* Copy in solution */
02338         Bvec_copy(vec, u);
02339         /* Add dirichlet condition */
02340         Bvec_axpy(vec, u_d, 1.0);
02341         /* Load up ions */
02342         ichop = 0;
02343         for (i=0; i<Gem_numVV(gm); i++) {
02344             val = 0;
02345             for (j=0; j<pbe->numIon; j++) {
02346                 q = pbe->ionQ[j];
02347                 conc = pbe->ionConc[j];
02348                 if (thee->type == PBE_NPBE || thee->type == PBE_SMPBE /* SMPB
E Added */) {
02349                     val += (q*conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));
02350                 } else if (thee->type == PBE_LPBE) {

```

```

02351                     val += (q*conc*(1 - q*Bvec_val(vec, i)));
02352                 }
02353             }
02354         Bvec_set(vec, i, val);
02355     }
02356     break;
02357
02358 case VDT_DIELX:
02359     Vnm_print(2, "Vfetk_fillArray: can't write out x-shifted diel!\n");
02360     return 0;
02361     break;
02362
02363 case VDT_DIELY:
02364     Vnm_print(2, "Vfetk_fillArray: can't write out y-shifted diel!\n");
02365     return 0;
02366     break;
02367
02368 case VDT_DIELZ:
02369     Vnm_print(2, "Vfetk_fillArray: can't write out z-shifted diel!\n");
02370     return 0;
02371     break;
02372
02373 case VDT_KAPPA:
02374     Vnm_print(2, "Vfetk_fillArray: can't write out kappa!\n");
02375     return 0;
02376     break;
02377
02378 default:
02379     Vnm_print(2, "Vfetk_fillArray: invalid data type (%d)!\n", type);
02380     return 0;
02381     break;
02382 }
02383
02384 return 1;
02385 }
02386
02387 VPUBLIC int Vfetk_write(Vfetk *thee, const char *iodev, const char *iofmt,
02388 const char *thost, const char *fname, Bvec *vec, Vdata_Format format) {
02389
02390     int i, j, ichop;
02391     Aprx *aprx;
02392     Gem *gm;
02393     Vio *sock;
02394
02395     VASSERT(thee != VNULL);
02396     aprx = thee->aprx;
02397     gm = thee->gm;
02398
02399     sock = Vio_ctor(iodev, iofmt, thost, fname, "w");
02400     if (sock == VNULL) {
02401         Vnm_print(2, "Vfetk_write: Problem opening virtual socket %s\n",
02402                 fname);
02403         return 0;
02404     }
02405     if (Vio_connect(sock, 0) < 0) {
02406         Vnm_print(2, "Vfetk_write: Problem connecting to virtual socket %s\n",
02407                 fname);

```

```

02408     return 0;
02409 }
02410
02411 /* Make sure vec has enough rows to accomodate the vertex data */
02412 if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02413     Vnm_print(2, "Vfetk_fillArray: insufficient space in Bvec!\n");
02414     Vnm_print(2, "Vfetk_fillArray: Have %d, need %d!\n", Bvec_numRB(vec, 0),
02415               Gem_numVV(gm));
02416     return 0;
02417 }
02418
02419 switch (format) {
02420
02421     case VDF_DX:
02422         Aprx_writeSOL(aprx, sock, vec, "DX");
02423         break;
02424     case VDF_AVIS:
02425         Aprx_writeSOL(aprx, sock, vec, "UCD");
02426         break;
02427     case VDF_UHBD:
02428         Vnm_print(2, "Vfetk_write: UHBD format not supported!\n");
02429         return 0;
02430     default:
02431         Vnm_print(2, "Vfetk_write: Invalid data format (%d)!\n", format);
02432         return 0;
02433 }
02434
02435
02436 Vio_connectFree(sock);
02437 Vio_dtor(&sock);
02438
02439 return 1;
02440 }
02441
02442 #endif

```

10.19 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/vpee.c File Reference

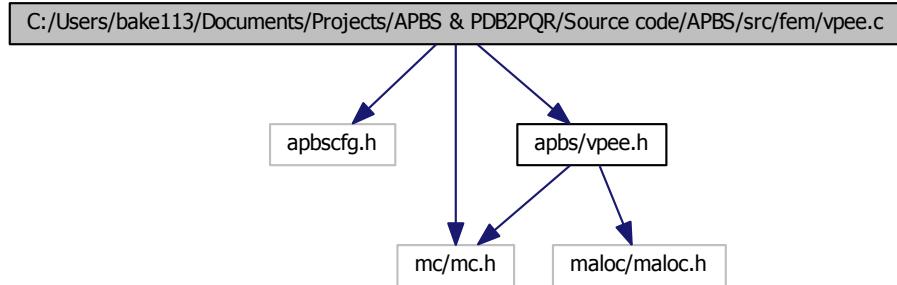
Class Vpee methods.

```

#include "apbscfg.h"
#include "mc/mc.h"
#include "apbs/vpee.h"

```

Include dependency graph for vpee.c:



Functions

- VPRIVATE int **Vpee_userDefined** (**Vpee** *thee, SS *sm)
- VPRIVATE int **Vpee_ourSimp** (**Vpee** *thee, SS *sm, int rcol)
- VEXTERNC double **Aprx_estNonlinResid** (Aprx *thee, SS *sm, Bvec *u, Bvec *ud, Bvec *f)
- VEXTERNC double **Aprx_estLocalProblem** (Aprx *thee, SS *sm, Bvec *u, Bvec *ud, Bvec *f)
- VEXTERNC double **Aprx_estDualProblem** (Aprx *thee, SS *sm, Bvec *u, Bvec *ud, Bvec *f)
- VPUBLIC **Vpee** * **Vpee_ctor** (Gem *gm, int localPartID, int killFlag, double killParam)

Construct the Vpee object.
- VPUBLIC int **Vpee_ctor2** (**Vpee** *thee, Gem *gm, int localPartID, int killFlag, double killParam)

FORTRAN stub to construct the Vpee object.
- VPUBLIC void **Vpee_dtor** (**Vpee** **thee)

Object destructor.
- VPUBLIC void **Vpee_dtor2** (**Vpee** *thee)

FORTRAN stub object destructor.
- VPUBLIC int **Vpee_markRefine** (**Vpee** *thee, AM *am, int level, int akey, int rcol, double etol, int bkey)

Mark simplices for refinement based on attenuated error estimates.
- VPUBLIC int **Vpee_numSS** (**Vpee** *thee)

Returns the number of simplices in the local partition.

10.19.1 Detailed Description

Class Vpee methods.

Author

Nathan Baker

Version

Id:

[vpee.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vpee.c](#).

10.20 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/fem/vpee.c

```

00001
00058 #include "apbscfg.h"
00059
00060 #if defined(HAVE_MC_H)
00061 #if defined(HAVE_MCX_H)
00062
00063 #include "mc/mc.h"
00064 #include "apbs/vpee.h"
00065
00066 VPRIVATE int Vpee_userDefined(Vpee *thee, SS *sm);
00067 VPRIVATE int Vpee_ourSimp(Vpee *thee, SS *sm, int rcol);
00068 VEXTERNC double Aprx_estNonlinResid(Aprx *thee, SS *sm,
00069     Bvec *u, Bvec *ud, Bvec *f);
00070 VEXTERNC double Aprx_estLocalProblem(Aprx *thee, SS *sm,
00071     Bvec *u, Bvec *ud, Bvec *f);
00072 VEXTERNC double Aprx_estDualProblem(Aprx *thee, SS *sm,
00073     Bvec *u, Bvec *ud, Bvec *f);
00074
00075 /* //////////////////////////////// */
00076 // Class Vpee: Non-inlineable methods
00077
00078
00079 /* //////////////////////////////// */
00080 // Routine: Vpee_ctor
00081 //
00082 // Author: Nathan Baker
00083 VPUBLIC Vpee* Vpee_ctor(Gem *gm, int localPartID, int killFlag, double
00084     killParam) {
00085
00086     Vpee *thee = VNULL;
00088
00089     /* Set up the structure */
00090     thee = Vmem_malloc(VNULL, 1, sizeof(Vpee));
00091     VASSERT( thee != VNULL );
00092     VASSERT( Vpee_ctor2(thee, gm, localPartID, killFlag, killParam) );
00093
00094     return thee;
00095 }
00096
00097 /* //////////////////////////////// */

```

```

00098 // Routine: Vpee_ctor2
00099 //
00100 // Author: Nathan Baker
00102 VPUBLIC int Vpee_ctor2(Vpee *thee, Gem *gm, int localPartID, int killFlag,
00103     double killParam) {
00104
00105     int invert, nLocalVerts;
00106     SS *simp;
00107     VV *vert;
00108     double radius, dx, dy, dz;
00109
00110     VASSERT(thee != VNULL);
00111
00112     /* Sanity check on input values */
00113     if (killFlag == 0) {
00114         Vnm_print(0, "Vpee_ctor2: No error attenuation outside partition.\n");
00115     } else if (killFlag == 1) {
00116         Vnm_print(0, "Vpee_ctor2: Error outside local partition ignored.\n");
00117     } else if (killFlag == 2) {
00118         Vnm_print(0, "Vpee_ctor2: Error ignored outside sphere with radius %4.3f
times the radius of the circumscribing sphere\n", killParam);
00119         if (killParam < 1.0) {
00120             Vnm_print(2, "Vpee_ctor2: Warning! Parameter killParam = %4.3 < 1.0!\n"
00121
00122                 killParam);
00123             Vnm_print(2, "Vpee_ctor2: This may result in non-optimal marking and re
finement!\n");
00124         }
00125     } else if (killFlag == 3) {
00126         Vnm_print(0, "Vpee_ctor2: Error outside local partition and immediate nei
ghbors ignored [NOT IMPLEMENTED].\n");
00127     } else {
00128         Vnm_print(2, "Vpee_ctor2: UNRECOGNIZED killFlag PARAMETER! BAILING!.n");
00129
00130         VASSERT(0);
00131     }
00132
00133     thee->gm = gm;
00134     thee->localPartID = localPartID;
00135     thee->killFlag = killFlag;
00136     thee->killParam = killParam;
00137     thee->mem = Vmem_ctor("APBS::VPEE");
00138
00139     /* Now, figure out the center of geometry for the local partition. The
00140      * general plan is to loop through the vertices, loop through the
00141      * vertices' simplex lists and find the vertices with simplices containing
00142      * chart values we're interested in. */
00143     thee->localPartCenter[0] = 0.0;
00144     thee->localPartCenter[1] = 0.0;
00145     thee->localPartCenter[2] = 0.0;
00146     nLocalVerts = 0;
00147     for (invert=0; invert<Gem_numVV(thee->gm); invert++) {
00148         vert = Gem_VV(thee->gm, invert);
00149         simp = VV_firstSS(vert);
00150         VASSERT(simp != VNULL);
00151         while (simp != VNULL) {
00152             if (SS_chart(simp) == thee->localPartID) {

```

```

00151     thee->localPartCenter[0] += VV_coord(vert, 0);
00152     thee->localPartCenter[1] += VV_coord(vert, 1);
00153     thee->localPartCenter[2] += VV_coord(vert, 2);
00154     nLocalVerts++;
00155     break;
00156   }
00157   simp = SS_link(simp, vert);
00158 }
00159 }
00160 VASSERT(nLocalVerts > 0);
00161 thee->localPartCenter[0] =
00162   thee->localPartCenter[0]/((double)(nLocalVerts));
00163 thee->localPartCenter[1] =
00164   thee->localPartCenter[1]/((double)(nLocalVerts));
00165 thee->localPartCenter[2] =
00166   thee->localPartCenter[2]/((double)(nLocalVerts));
00167 Vnm_print(0, "Vpee_ctor2: Part %d centered at (%4.3f, %4.3f, %4.3f)\n",
00168   thee->localPartID, thee->localPartCenter[0], thee->localPartCenter[1],
00169   thee->localPartCenter[2]);
00170
00171 /* Now, figure out the radius of the sphere circumscribing the local
00172 * partition. We need to keep track of vertices so we don't double count
00173 * them. */
00174 thee->localPartRadius = 0.0;
00175 for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
00176   vert = Gem_VV(thee->gm, ivert);
00177   simp = VV_firstSS(vert);
00178   VASSERT(simp != VNULL);
00179   while (simp != VNULL) {
00180     if (SS_chart(simp) == thee->localPartID) {
00181       dx = thee->localPartCenter[0] - VV_coord(vert, 0);
00182       dy = thee->localPartCenter[1] - VV_coord(vert, 1);
00183       dz = thee->localPartCenter[2] - VV_coord(vert, 2);
00184       radius = dx*dx + dy*dy + dz*dz;
00185       if (radius > thee->localPartRadius) thee->localPartRadius =
00186         radius;
00187       break;
00188     }
00189     simp = SS_link(simp, vert);
00190   }
00191 }
00192 thee->localPartRadius = VSQRT(thee->localPartRadius);
00193 Vnm_print(0, "Vpee_ctor2: Part %d has circumscribing sphere of radius %4.3f\n"
00194   ,
00195   thee->localPartID, thee->localPartRadius);
00196
00197 return 1;
00198 }
00199
00200 /* //////////////////////////////// */
00201 // Routine: Vpee_dtor
00202 //
00203 // Author: Nathan Baker
00204 VPUBLIC void Vpee_dtor(Vpee **thee) {
00205   if ((*thee) != VNULL) {

```

```

00208     Vpee_dtor2(*thee);
00209     Vmem_free(VNULL, 1, sizeof(Vpee), (void **)thee);
00210     (*thee) = VNULL;
00211 }
00212
00213 }
00214
00215 /* //////////////////////////////// */
00216 // Routine: Vpee_dtor2
00217 //
00218 // Author: Nathan Baker
00219 VPUBLIC void Vpee_dtor2(Vpee *thee) { Vmem_dtor(&(thee->mem)); }
00220
00221 /* //////////////////////////////// */
00222 // Routine: Vpee_markRefine
00223 //
00224 //
00225 // Author: Nathan Baker (and Michael Holst: the author of AM_markRefine, on
00226 // which this is based)
00227 VPUBLIC int Vpee_markRefine(Vpee *thee, AM *am, int level, int akey, int rcol,
00228     double etol, int bkey) {
00229
00230     Aprx *aprx;
00231     int marked = 0;
00232     int markMe, i, smid, count, currentQ;
00233     double minError = 0.0;
00234     double maxError = 0.0;
00235     double errEst = 0.0;
00236     double mlevel, barrier;
00237     SS *sm;
00238
00239
00240     VASSERT(thee != VNULL);
00241
00242     /* Get the Aprx object from AM */
00243     aprx = am->aprx;
00244
00245     /* input check and some i/o */
00246     if ( ! ((-1 <= akey) && (akey <= 4)) ) {
00247         Vnm_print(0,"Vpee_markRefine: bad refine key; simplices marked = %d\n",
00248                 marked);
00249         return marked;
00250     }
00251
00252     /* For uniform markings, we have no effect */
00253     if ((-1 <= akey) && (akey <= 0)) {
00254         marked = Gem_markRefine(thee->gm, akey, rcol);
00255         return marked;
00256     }
00257
00258     /* Informative I/O */
00259     if (akey == 2) {
00260         Vnm_print(0,"Vpee_estRefine: using Aprx_estNonlinResid().\n");
00261     } else if (akey == 3) {
00262         Vnm_print(0,"Vpee_estRefine: using Aprx_estLocalProblem().\n");
00263     } else if (akey == 4) {
00264         Vnm_print(0,"Vpee_estRefine: using Aprx_estDualProblem().\n");
00265     } else {
00266

```

```

00267     Vnm_print(0, "Vpee_estRefine: bad key given; simplices marked = %d\n",
00268             marked);
00269     return marked;
00270 }
00271 if (thee->killFlag == 0) {
00272     Vnm_print(0, "Vpee_markRefine: No error attenuation -- simplices in all p
artitions will be marked.\n");
00273 } else if (thee->killFlag == 1) {
00274     Vnm_print(0, "Vpee_markRefine: Maximum error attenuation -- only simplice
s in local partition will be marked.\n");
00275 } else if (thee->killFlag == 2) {
00276     Vnm_print(0, "Vpee_markRefine: Spherical error attenuutation -- simplices
within a sphere of %4.3f times the size of the partition will be marked\n",
00277             thee->killParam);
00278 } else if (thee->killFlag == 2) {
00279     Vnm_print(0, "Vpee_markRefine: Neighbor-based error attenuation -- simpli
ces in the local and neighboring partitions will be marked [NOT IMPLEMENTED]!\n")
;
00280     VASSERT(0);
00281 } else {
00282     Vnm_print(2, "Vpee_markRefine: bogus killFlag given; simplices marked = %d
\n",
00283             marked);
00284     return marked;
00285 }
00286 /* set the barrier type */
00287 mlevel = (etol*etol) / Gem_numSS(thee->gm);
00288 if (bkey == 0) {
00289     barrier = (etol*etol);
00290     Vnm_print(0, "Vpee_estRefine: forcing [err per S] < [TOL] = %g\n",
00291             barrier);
00292 } else if (bkey == 1) {
00293     barrier = mlevel;
00294     Vnm_print(0, "Vpee_estRefine: forcing [err per S] < [(TOL^2/numS)^{1/2}] =
%g\n",
00295             VSQRT(barrier));
00296 } else {
00297     Vnm_print(0, "Vpee_estRefine: bad bkey given; simplices marked = %d\n",
00298             marked);
00299     return marked;
00300 }
00301 }
00302 /* timer */
00303 Vnm_tstart(30, "error estimation");
00305
00306 /* count = num generations to produce from marked simplices (minimally) */
00307 count = 1; /* must be >= 1 */
00308
00309 /* check the refinement Q for emptiness */
00310 currentQ = 0;
00311 if (Gem_numSQ(thee->gm, currentQ) > 0) {
00312     Vnm_print(0, "Vpee_markRefine: non-empty refinement Q%d....clearing..",
00313             currentQ);
00314     Gem_resetSQ(thee->gm, currentQ);
00315     Vnm_print(0, "..done.\n");
00316 }

```

```

00317     if (Gem_numSQ(thee->gm,!currentQ) > 0) {
00318         Vnm_print(0,"Vpee_markRefine: non-empty refinement Q%d....clearing..",
00319                     !currentQ);
00320         Gem_resetSQ(thee->gm,!currentQ);
00321         Vnm_print(0,"..done.\n");
00322     }
00323     VASSERT( Gem_numSQ(thee->gm,currentQ) == 0 );
00324     VASSERT( Gem_numSQ(thee->gm,!currentQ) == 0 );
00325
00326     /* clear everyone's refinement flags */
00327     Vnm_print(0,"Vpee_markRefine: clearing all simplex refinement flags..");
00328     for (i=0; i<Gem_numSS(thee->gm); i++) {
00329         if ( (i>0) && (i % VPRTKEY) == 0 ) Vnm_print(0, "[MS:%d]",i);
00330         sm = Gem_SS(thee->gm,i);
00331         SS_setRefineKey(sm,currentQ,0);
00332         SS_setRefineKey(sm,!currentQ,0);
00333         SS_setRefinementCount(sm,0);
00334     }
00335     Vnm_print(0,"..done.\n");
00336
00337     /* NON-ERROR-BASED METHODS */
00338     /* Simplex flag clearing */
00339     if (akey == -1) return marked;
00340     /* Uniform & user-defined refinement*/
00341     if ((akey == 0) || (akey == 1)) {
00342         smid = 0;
00343         while ( smid < Gem_numSS(thee->gm) ) {
00344             /* Get the simplex and find out if it's markable */
00345             sm = Gem_SS(thee->gm,smid);
00346             markMe = Vpee_ourSimp(thee, sm, rcol);
00347             if (markMe) {
00348                 if (akey == 0) {
00349                     marked++;
00350                     Gem_appendSQ(thee->gm,currentQ, sm);
00351                     SS_setRefineKey(sm,currentQ,1);
00352                     SS_setRefinementCount(sm,count);
00353                 } else if (Vpee_userDefined(thee, sm)) {
00354                     marked++;
00355                     Gem_appendSQ(thee->gm,currentQ, sm);
00356                     SS_setRefineKey(sm,currentQ,1);
00357                     SS_setRefinementCount(sm,count);
00358                 }
00359             }
00360             smid++;
00361         }
00362     }
00363
00364     /* ERROR-BASED METHODS */
00365     /* gerror = global error accumulation */
00366     aprx->gerror = 0.;
00367
00368     /* traverse the simplices and process the error estimates */
00369     Vnm_print(0,"Vpee_markRefine: estimating error..");
00370     smid = 0;
00371     while ( smid < Gem_numSS(thee->gm) ) {
00372         /* Get the simplex and find out if it's markable */

```

```

00374     sm = Gem_SS(thee->gm,smid);
00375     markMe = Vpee_ourSimp(thee, sm, rcol);
00376
00377     if ( (smid>0) && (smid % VPRTKEY) == 0 ) Vnm_print(0, "[MS:%d]",smid);
00378
00379     /* Produce an error estimate for this element if it is in the set */
00380     if (markMe) {
00381         if (akey == 2) {
00382             errEst = Aprx_estNonlinResid(aprx, sm, am->u,am->ud,am->f);
00383         } else if (akey == 3) {
00384             errEst = Aprx_estLocalProblem(aprx, sm, am->u,am->ud,am->f);
00385         } else if (akey == 4) {
00386             errEst = Aprx_estDualProblem(aprx, sm, am->u,am->ud,am->f);
00387         }
00388         VASSERT( errEst >= 0. );
00389
00390         /* if error estimate above tol, mark element for refinement */
00391         if ( errEst > barrier ) {
00392             marked++;
00393             Gem_appendSQ(thee->gm,currentQ, sm); /*add to refinement Q*/
00394             SS_setRefineKey(sm,currentQ,1); /* note now on refine Q */
00395             SS_setRefinementCount(sm,count); /* refine X many times? */
00396         }
00397
00398         /* keep track of min/max errors over the mesh */
00399         minError = VMIN2( VSQRT(VABS(errEst)), minError );
00400         maxError = VMAX2( VSQRT(VABS(errEst)), maxError );
00401
00402         /* store the estimate */
00403         Bvec_set( aprx->wev, smid, errEst );
00404
00405         /* accumulate into global error (errEst is SQUARED already) */
00406         aprx->gerror += errEst;
00407
00408         /* otherwise store a zero for the estimate */
00409         } else {
00410             Bvec_set( aprx->wev, smid, 0. );
00411         }
00412
00413         smid++;
00414     }
00415
00416     /* do some i/o */
00417     Vnm_print(0, "...done. [marked=<%d/%d>]\n",marked,Gem_numSS(thee->gm));
00418     Vnm_print(0, "Vpee_estRefine: TOL=<%g> Global_Error=<%g>\n",
00419                etol, aprx->gerror);
00420     Vnm_print(0, "Vpee_estRefine: (TOL^2/numS)^(1/2)=<%g> Max_Ele_Error=<%g>\n",
00421                VSQRT(mlevel),maxError);
00422     Vnm_tstop(30, "error estimation");
00423
00424     /* check for making the error tolerance */
00425     if ((bkey == 1) && (aprx->gerror <= etol)) {
00426         Vnm_print(0,
00427                    "Vpee_estRefine: *****\n");
00428         Vnm_print(0,
00429                    "Vpee_estRefine: Global Error criterion met; setting marked=0.\n");
00430         Vnm_print(0,

```

```

00431         "Vpee_estRefine: *****\n");
00432     marked = 0;
00433 }
00434
00435
00436 /* return */
00437 return marked;
00438
00439 }
00440
00441 /* /////////////////////////////////
00442 // Routine: Vpee_numSS
00443 //
00444 // Author: Nathan Baker
00445 VPUBLIC int Vpee_numSS(Vpee *thee) {
00446     int num = 0;
00447     int isimp;
00448
00449     for (isimp=0; isimp<Gem_numSS(thee->gm); isimp++) {
00450         if (SS_chart(Gem_SS(thee->gm, isimp)) == thee->localPartID) num++;
00451     }
00452
00453     return num;
00454 }
00455
00456
00457 /* /////////////////////////////////
00458 // Routine: Vpee_userDefined
00459 //
00460 // Purpose: Reduce code bloat by wrapping up the common steps for getting the
00461 // user-defined error estimate
00462 //
00463 // Author: Nathan Baker
00464 VPRIVATE int Vpee_userDefined(Vpee *thee, SS *sm) {
00465
00466     int invert, icoord, chart[4], fType[4], vType[4];
00467     double vx[4][3];
00468
00469     for (invert=0; invert<Gem_dimVV(thee->gm); invert++) {
00470         fType[invert] = SS_faceType(sm,invert);
00471         vType[invert] = VV_type(SS_vertex(sm,invert) );
00472         chart[invert] = VV_chart(SS_vertex(sm,invert) );
00473         for (icoord=0; icoord<Gem_dimII(thee->gm); icoord++) {
00474             vx[invert][icoord] = VV_coord(SS_vertex(sm,invert), icoord );
00475         }
00476     }
00477
00478     return thee->gm->pde->markSimplex(Gem_dim(thee->gm), Gem_dimII(thee->gm),
00479                                         SS_type(sm), fType, vType, chart, vx, sm);
00480 }
00481
00482 /* /////////////////////////////////
00483 // Routine: Vpee_ourSimp
00484 //
00485 // Purpose: Reduce code bloat by wrapping up the common steps for determining
00486 // whether the given simplex can be marked (i.e., belongs to our
00487 // partition or overlap region)
00488 //
00489 // Returns: 1 if could be marked, 0 otherwise

```

```

00490 //
00491 // Author: Nathan Baker
00493 VPRIVATE int Vpee_ourSimp(Vpee *thee, SS *sm, int rcol) {
00494
00495     int invert;
00496     double dist, dx, dy, dz;
00497
00498     if (thee->killFlag == 0) return 1;
00499     else if (thee->killFlag == 1) {
00500         if ((SS_chart(sm) == rcol) || (rcol < 0)) return 1;
00501     } else if (thee->killFlag == 2) {
00502         if (rcol < 0) return 1;
00503     } else {
00504         /* We can only do distance-based searches on the local partition */
00505         VASSERT(rcol == thee->localPartID);
00506         /* Find the closest distance between this simplex and the
00507             * center of the local partition and check it against
00508             * (thee->localPartRadius*thee->killParam) */
00509         dist = 0;
00510         for (invert=0; invert<SS_dimVV(sm); invert++) {
00511             dx = VV_coord(SS_vertex(sm, invert), 0) -
00512                 thee->localPartCenter[0];
00513             dy = VV_coord(SS_vertex(sm, invert), 1) -
00514                 thee->localPartCenter[1];
00515             dz = VV_coord(SS_vertex(sm, invert), 2) -
00516                 thee->localPartCenter[2];
00517             dist = VSQRT((dx*dx + dy*dy + dz*dz));
00518         }
00519         if (dist < thee->localPartRadius*thee->killParam) return 1;
00520     }
00521 } else if (thee->killFlag == 3) VASSERT(0);
00522 else VASSERT(0);
00523
00524     return 0;
00525
00526 }
00527
00528 #endif
00529 #endif

```

10.21 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/femparm.h File Reference

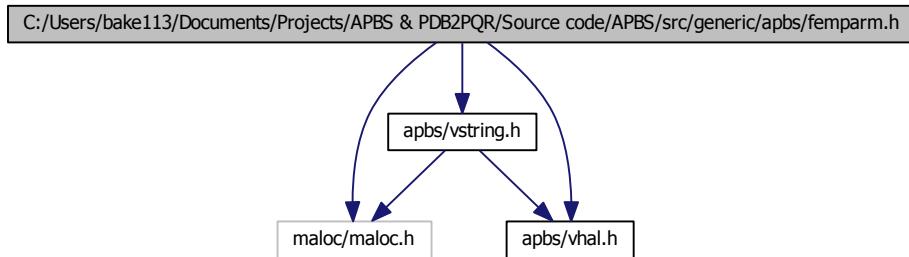
Contains declarations for class APOLparm.

```

#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vstring.h"

```

Include dependency graph for femparm.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sFEMparm`

Parameter structure for FEM-specific variables from input files.

TypeDefs

- typedef enum `eFEMparm_EtolType` `FEMparm_EtolType`
Declare FEmparm_EtolType type.
- typedef enum `eFEMparm_EstType` `FEMparm_EstType`
Declare FEmparm_EstType type.
- typedef enum `eFEMparm_CalcType` `FEMparm_CalcType`
Declare FEmparm_CalcType type.
- typedef struct `sFEMparm` `FEMparm`
Declaration of the FEMparm class as the FEMparm structure.

Enumerations

- enum eFEMparm_EtolType { **FET_SIMP** = 0, **FET_GLOB** = 1, **FET_FRAC** = 2 }

Adaptive refinement error estimate tolerance key.

- enum eFEMparm_EstType {

FRT_UNIF = 0, **FRT_GEOM** = 1, **FRT_RESI** = 2, **FRT_DUAL** = 3, **FRT_LOCA** = 4 }

Adaptive refinement error estimator method.

- enum eFEMparm_CalcType { **FCT_MANUAL**, **FCT_NONE** }

Calculation type.

Functions

- VEXTERNC **FEMparm** * **FEMparm_ctor** (**FEMparm_CalcType** type)

Construct FEMparm.

- VEXTERNC int **FEMparm_ctor2** (**FEMparm** *thee, **FEMparm_CalcType** type)

FORTRAN stub to construct FEMparm.

- VEXTERNC void **FEMparm_dtor** (**FEMparm** **thee)

Object destructor.

- VEXTERNC void **FEMparm_dtor2** (**FEMparm** *thee)

FORTRAN stub for object destructor.

- VEXTERNC int **FEMparm_check** (**FEMparm** *thee)

Consistency check for parameter values stored in object.

- VEXTERNC void **FEMparm_copy** (**FEMparm** *thee, **FEMparm** *source)

Copy target object into thee.

- VEXTERNC Vrc_Codes **FEMparm_parseToken** (**FEMparm** *thee, char tok[VMAX_-BUFSIZE], Vio *sock)

Parse an MG keyword from an input file.

10.21.1 Detailed Description

Contains declarations for class APOLparm. Contains declarations for class FEMparm.

Version

Id:

[apolparm.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Version

Id:

[femparm.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [femparm.h](#).

10.22 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/femparm.h

```
00001
00063 #ifndef _FEMPARM_H_
00064 #define _FEMPARM_H_
00065
00066 /* Generic header files */
00067 #include "maloc/maloc.h"
00068 #include "apbs/vhal.h"
00069 #include "apbs/vstring.h"
00070
00076 enum eFEMParm_EtolType {
00077     FET_SIMP=0,
00078     FET_GLOB=1,
00079     FET_FRAC=2
00080 };
00081
00087 typedef enum eFEMParm_EtolType FEMParm_EtolType;
00088
00095 enum eFEMParm_EstType {
00096     FRT_UNIF=0,
00097     FRT_GEOM=1,
00098     FRT_RESI=2,
00099     FRT_DUAL=3,
00101     FRT_LOCA=4
00102 };
00103
00108 typedef enum eFEMParm_EstType FEMParm_EstType;
00109
00114 enum eFEMParm_CalcType {
00115     FCT_MANUAL,
00116     FCT_NONE
00117 };
00118
00123 typedef enum eFEMParm_CalcType FEMParm_CalcType;
00124
00130 struct sFEMParm {
00131
00132     int parsed;
00135     FEMParm_CalcType type;
00136     int settype;
00137     double glen[3];
00138     int setglen;
00139     double etol;
00141     int setetol;
00142     FEMParm_EtolType ekey;
00144     int setekey;
00145     FEMParm_EstType akeyPRE;
00148     int setakeyPRE;
00149     FEMParm_EstType akeySOLVE;
00151     int setakeySOLVE;
00152     int targetNum;
00156     int settargetNum;
00157     double targetRes;
00161     int settargetRes;
```

```

00162     int maxsolve;
00163     int setmaxsolve;
00164     int maxvert;
00166     int setmaxvert;
00167     int pkey;
00170     int useMesh;
00171     int meshID;
00173 };
00174
00179 typedef struct sFEMparm FEMparm;
00180
00181 /* //////////////////////////////// */
00182 // Class Nosh: Non-inlineable methods (nosh.c)
00184
00191 VEXTERNC FEMparm* FEMparm_ctor(FEMparm_CalcType type);
00192
00200 VEXTERNC int FEMparm_ctor2(FEMparm *thee, FEMparm_CalcType type);
00201
00207 VEXTERNC void FEMparm_dtor(FEMparm **thee);
00208
00214 VEXTERNC void FEMparm_dtor2(FEMparm *thee);
00215
00223 VEXTERNC int FEMparm_check(FEMparm *thee);
00224
00231 VEXTERNC void FEMparm_copy(FEMparm *thee, FEMparm *source);
00232
00243 VEXTERNC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
00244   Vio *sock);
00245
00246 #endif
00247

```

10.23 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/mgparm.h File Reference

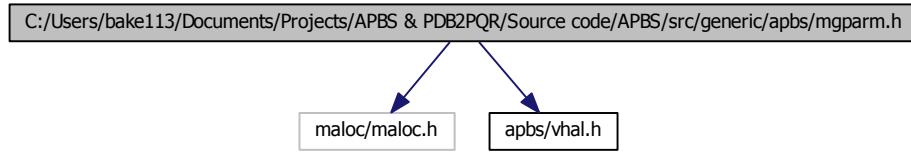
Contains declarations for class MGparm.

```

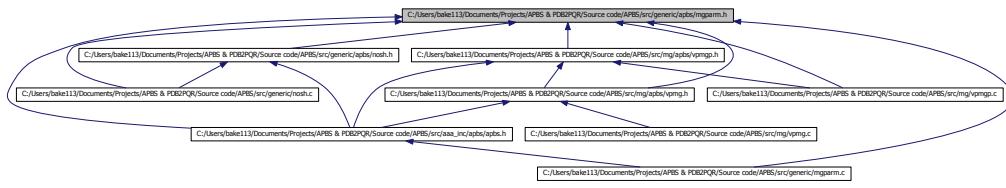
#include "malloc/malloc.h"
#include "apbs/vhal.h"

```

Include dependency graph for mgparm.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **sMGparm**

Parameter structure for MG-specific variables from input files.

Typedefs

- typedef enum **eMGparm_CalcType** **MGparm_CalcType**
Declare MGparm_CalcType type.
- typedef enum **eMGparm_CentMeth** **MGparm_CentMeth**
Declare MGparm_CentMeth type.
- typedef struct **sMGparm** **MGparm**
Declaration of the MGparm class as the MGparm structure.

Enumerations

- enum **eMGparm_CalcType** {

```
MCT_MANUAL = 0, MCT_AUTO = 1, MCT_PARALLEL = 2, MCT_DUMMY = 3,
MCT_NONE = 4 }
```

Calculation type.

- enum eMgparm_CentMeth { MCM_POINT = 0, MCM_MOLECULE = 1, MCM_FOCUS = 2 }

Centering method.

Functions

- VEXTERNC int MGparm_getNx (MGparm *thee)
Get number of grid points in x direction.
- VEXTERNC int MGparm_getNy (MGparm *thee)
Get number of grid points in y direction.
- VEXTERNC int MGparm_getNz (MGparm *thee)
Get number of grid points in z direction.
- VEXTERNC double MGparm_getHx (MGparm *thee)
Get grid spacing in x direction (Å)
- VEXTERNC double MGparm_getHy (MGparm *thee)
Get grid spacing in y direction (Å)
- VEXTERNC double MGparm_getHz (MGparm *thee)
Get grid spacing in z direction (Å)
- VEXTERNC void MGparm_setCenterX (MGparm *thee, double x)
Set center x-coordinate.
- VEXTERNC void MGparm_setCenterY (MGparm *thee, double y)
Set center y-coordinate.
- VEXTERNC void MGparm_setCenterZ (MGparm *thee, double z)
Set center z-coordinate.
- VEXTERNC double MGparm_getCenterX (MGparm *thee)
Get center x-coordinate.
- VEXTERNC double MGparm_getCenterY (MGparm *thee)
Get center y-coordinate.
- VEXTERNC double MGparm_getCenterZ (MGparm *thee)
Get center z-coordinate.
- VEXTERNC MGparm * MGparm_ctor (MGparm_CalcType type)
Construct MGparm object.
- VEXTERNC Vrc_Codes MGparm_ctor2 (MGparm *thee, MGparm_CalcType type)

FORTRAN stub to construct MGparm object.
- VEXTERNC void MGparm_dtor (MGparm **thee)

Object destructor.

- VEXTERNC void [MGparm_dtor2](#) ([MGparm](#) *thee)
FORTRAN stub for object destructor.
- VEXTERNC Vrc_Codes [MGparm_check](#) ([MGparm](#) *thee)
Consistency check for parameter values stored in object.
- VEXTERNC void [MGparm_copy](#) ([MGparm](#) *thee, [MGparm](#) *parm)
Copy MGparm object into thee.
- VEXTERNC Vrc_Codes [MGparm_parseToken](#) ([MGparm](#) *thee, char tok[VMAX_-BUFSIZE], Vio *sock)
Parse an MG keyword from an input file.

10.23.1 Detailed Description

Contains declarations for class MGparm.

Version

Id:

[mgparm.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*
```

```

*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [mgparm.h](#).

10.24 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/mgparm.h

```

00001
00064 #ifndef _MGPARM_H_
00065 #define _MGPARM_H_
00066
00067 #include "maloc/maloc.h"
00068 #include "apbs/vhal.h"
00069
00074 enum eMGParm_CalcType {
00075     MCT_MANUAL=0,
00076     MCT_AUTO=1,
00077     MCT_PARALLEL=2,
00078     MCT_DUMMY=3,
00079     MCT_NONE=4
00080 };
00081
00086 typedef enum eMGParm_CalcType MGparm_CalcType;
00087
00092 enum eMGParm_CentMeth {
00093     MCM_POINT=0,
00094     MCM_MOLECULE=1,
00095     MCM_FOCUS=2
00096 };

```

```
00097
00102 typedef enum eMGparm_CentMeth MGparm_CentMeth;
00111 struct sMGparm {
00112
00113     MGparm_CalcType type;
00114     int parsed;
00116     /* *** GENERIC PARAMETERS *** */
00117     int dime[3];
00118     int setdime;
00119     Vchrg_Meth chgm;
00120     int setchgm;
00121     Vchrg_Src chgs;
00124     /* *** TYPE 0 PARAMETERS (SEQUENTIAL MANUAL) *** */
00125     int nlev;
00127     int setnlev;
00128     double etol;
00129     int setetol;
00130     double grid[3];
00131     int setgrid;
00132     double glen[3];
00133     int setglen;
00134     MGparm_CentMeth cmeth;
00135     double center[3];
00143     int centmol;
00146     int setgcent;
00148     /* ***** TYPE 1 & 2 PARAMETERS (SEQUENTIAL & PARALLEL AUTO-FOCUS) *** */
00149     double cglen[3];
00150     int setcglen;
00151     double fglen[3];
00152     int setfglen;
00153     MGparm_CentMeth ccmeth;
00154     double ccenter[3];
00155     int ccentmol;
00158     int setcgcent;
00159     MGparm_CentMeth fcmeth;
00160     double fcenter[3];
00161     int fcenmol;
00164     int setfgcent;
00167     /* ***** TYPE 2 PARAMETERS (PARALLEL AUTO-FOCUS) ***** */
00168     double partDisjCenter[3];
00170     double partDisjLength[3];
00172     int partDisjOwnSide[6];
00175     int pdime[3];
00176     int setpdime;
00177     int proc_rank;
00178     int setrank;
00179     int proc_size;
00180     int setsize;
00181     double ofrac;
00182     int setofrac;
00183     int async;
00184     int setasync;
00186     int nonlintype;
00187     int setnonlintype;
00189     int method;
00190     int setmethod;
00192     int useAqua;
```

```

00193 int setUseAqua;
00194 };
00195
00200 typedef struct sMGparm MGparm;
00201
00208 VEXTERNC int MGparm_getNx(MGparm *thee);
00209
00216 VEXTERNC int MGparm_getNy(MGparm *thee);
00217
00224 VEXTERNC int MGparm_getNz(MGparm *thee);
00225
00232 VEXTERNC double MGparm_getHx(MGparm *thee);
00233
00240 VEXTERNC double MGparm_getHy(MGparm *thee);
00241
00248 VEXTERNC double MGparm_getHz(MGparm *thee);
00249
00256 VEXTERNC void MGparm_setCenterX(MGparm *thee, double x);
00257
00264 VEXTERNC void MGparm_setCenterY(MGparm *thee, double y);
00265
00272 VEXTERNC void MGparm_setCenterZ(MGparm *thee, double z);
00273
00280 VEXTERNC double MGparm_getCenterX(MGparm *thee);
00281
00288 VEXTERNC double MGparm_getCenterY(MGparm *thee);
00289
00296 VEXTERNC double MGparm_getCenterZ(MGparm *thee);
00297
00304 VEXTERNC MGparm* MGparm_ctor(MGparm_CalcType type);
00305
00313 Vrc_Codes MGparm_ctor2(MGparm *thee, MGparm_CalcType type);
00314
00320 VEXTERNC void MGparm_dtor(MGparm **thee);
00321
00327 VEXTERNC void MGparm_dtor2(MGparm *thee);
00328
00335 VEXTERNC Vrc_Codes MGparm_check(MGparm *thee);
00336
00343 VEXTERNC void MGparm_copy(MGparm *thee, MGparm *parm);
00344
00354 VEXTERNC Vrc_Codes MGparm_parseToken(MGparm *thee, char tok[VMAX_BUFSIZE],
00355 Vio *sock);
00356
00357 #endif
00358

```

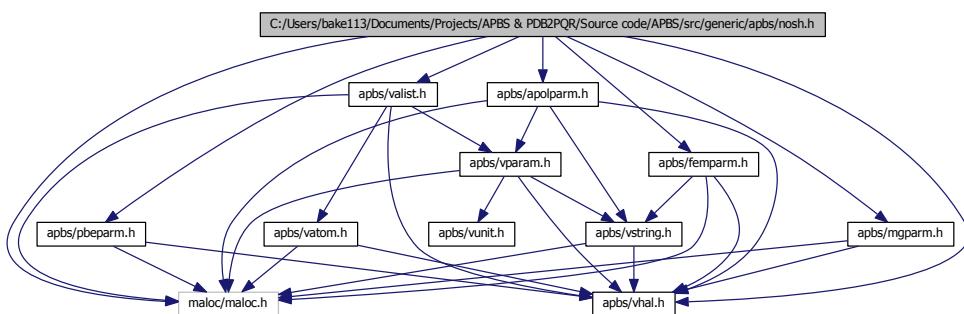
10.25 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/nosh.h File Reference

Contains declarations for class NOsh.

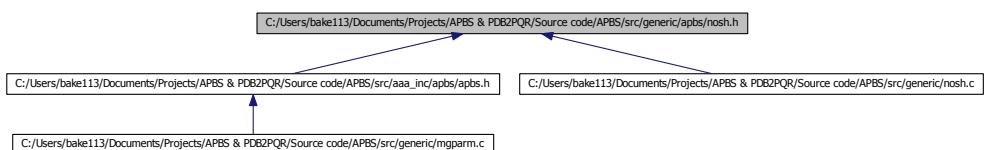
```
#include "maloc/maloc.h"
```

```
#include "apbs/vhal.h"
#include "apbs/pbeparm.h"
#include "apbs/mgparm.h"
#include "apbs/femparm.h"
#include "apbs/apolparm.h"
#include "apbs/valist.h"
```

Include dependency graph for nosh.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sNOsh_calc](#)
Calculation class for use when parsing fixed format input files.
- struct [sNOsh](#)
Class for parsing fixed format input files.

Defines

- `#define NOSH_MAXMOL 20`
Maximum number of molecules in a run.
- `#define NOSH_MAXCALC 20`
Maximum number of calculations in a run.
- `#define NOSH_MAXPRINT 20`
Maximum number of PRINT statements in a run.
- `#define NOSH_MAXPOP 20`
Maximum number of operations in a PRINT statement.

Typedefs

- `typedef enum eNOsh_MolFormat NOsh_MolFormat`
Declare NOsh_MolFormat type.
- `typedef enum eNOsh_CalcType NOsh_CalcType`
Declare NOsh_CalcType type.
- `typedef enum eNOsh_ParmFormat NOsh_ParmFormat`
Declare NOsh_ParmFormat type.
- `typedef enum eNOsh_PrintType NOsh_PrintType`
Declare NOsh_PrintType type.
- `typedef struct sNOsh_calc NOsh_calc`
Declaration of the NOsh_calc class as the NOsh_calc structure.
- `typedef struct sNOsh NOsh`
Declaration of the NOsh class as the NOsh structure.

Enumerations

- `enum eNOsh_MolFormat { NMF_PQR = 0, NMF_PDB = 1, NMF_XML = 2 }`
Molecule file format types.
- `enum eNOsh_CalcType { NCT_MG = 0, NCT_FEM = 1, NCT_APOL = 2 }`
NOsh calculation types.
- `enum eNOsh_ParmFormat { NPF_FLAT = 0, NPF_XML = 1 }`
Parameter file format types.
- `enum eNOsh_PrintType {`
 `NPT_ENERGY = 0, NPT_FORCE = 1, NPT_ELECENERGY, NPT_ELECFORCE,`
 `NPT_APOLENERGY, NPT_APOLFORCE }`
NOsh print types.

Functions

- VEXTERNC char * NOsh_getMolpath (NOsh *thee, int imol)
Returns path to specified molecule.
- VEXTERNC char * NOsh_getDielXpath (NOsh *thee, int imap)
Returns path to specified x-shifted dielectric map.
- VEXTERNC char * NOsh_getDielYpath (NOsh *thee, int imap)
Returns path to specified y-shifted dielectric map.
- VEXTERNC char * NOsh_getDielZpath (NOsh *thee, int imap)
Returns path to specified z-shifted dielectric map.
- VEXTERNC char * NOsh_getKappapath (NOsh *thee, int imap)
Returns path to specified kappa map.
- VEXTERNC char * NOsh_getPotpath (NOsh *thee, int imap)
Returns path to specified potential map.
- VEXTERNC char * NOsh_getChargepath (NOsh *thee, int imap)
Returns path to specified charge distribution map.
- VEXTERNC NOsh_calc * NOsh_getCalc (NOsh *thee, int icalc)
Returns specified calculation object.
- VEXTERNC int NOsh_getDielfmt (NOsh *thee, int imap)
Returns format of specified dielectric map.
- VEXTERNC int NOsh_getKappafmt (NOsh *thee, int imap)
Returns format of specified kappa map.
- VEXTERNC int NOsh_getPotfmt (NOsh *thee, int imap)
Returns format of specified potential map.
- VEXTERNC int NOsh_getChargefmt (NOsh *thee, int imap)
Returns format of specified charge map.
- VEXTERNC NOsh_PrintType NOsh_printWhat (NOsh *thee, int iprint)
Return an integer ID of the observable to print .
- VEXTERNC char * NOsh_elecname (NOsh *thee, int ielec)
Return an integer mapping of an ELEC statement to a calculation ID .
- VEXTERNC int NOsh_elec2calc (NOsh *thee, int icalc)
Return the name of an elec statement.
- VEXTERNC int NOsh_apol2calc (NOsh *thee, int icalc)
Return the name of an apol statement.
- VEXTERNC int NOsh_printNarg (NOsh *thee, int iprint)
Return number of arguments to PRINT statement .
- VEXTERNC int NOsh_printOp (NOsh *thee, int iprint, int iarg)
Return integer ID for specified operation .
- VEXTERNC int NOsh_printCalc (NOsh *thee, int iprint, int iarg)
Return calculation ID for specified PRINT statement .

- VEXTERNC `NOsh *`[NOsh_ctor](#) (int rank, int size)
Construct NOsh.
- VEXTERNC `NOsh_calc *`[NOsh_calc_ctor](#) (`NOsh_CalcType` calcType)
Construct NOsh_calc.
- VEXTERNC int `NOsh_calc_copy` (`NOsh_calc *thee, NOsh_calc *source`)
Copy NOsh_calc object into thee.
- VEXTERNC void `NOsh_calc_dtor` (`NOsh_calc **thee`)
Object destructor.
- VEXTERNC int `NOsh_ctor2` (`NOsh *thee, int rank, int size`)
FORTRAN stub to construct NOsh.
- VEXTERNC void `NOsh_dtor` (`NOsh **thee`)
Object destructor.
- VEXTERNC void `NOsh_dtor2` (`NOsh *thee`)
FORTRAN stub for object destructor.
- VEXTERNC int `NOsh_parseInput` (`NOsh *thee, Vio *sock`)
Parse an input file from a socket.
- VEXTERNC int `NOsh_parseInputFile` (`NOsh *thee, char *filename`)
Parse an input file only from a file.
- VEXTERNC int `NOsh_setupElecCalc` (`NOsh *thee, Valist *alist[NOSH_MAXMOL]`)
Setup the series of electrostatics calculations.
- VEXTERNC int `NOsh_setupApolCalc` (`NOsh *thee, Valist *alist[NOSH_MAXMOL]`)
Setup the series of non-polar calculations.

10.25.1 Detailed Description

Contains declarations for class NOsh.

Version

Id:

[nosh.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file nosh.h.

10.26 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/nosh.h

```
00001
00062 #ifndef _NOSH_H_
00063 #define _NOSH_H_
00064
00065 /* Generic headers */
00066 #include "malloc/malloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Headers specific to this file */
00070 #include "apbs/pbeparm.h"
00071 #include "apbs/mgparm.h"
00072 #include "apbs/femparm.h"
00073 #include "apbs/apolparm.h"
00074 #include "apbs/valist.h"
00075
00078 #define NOSH_MAXMOL 20
00079
00082 #define NOSH_MAXCALC 20
00083
00086 #define NOSH_MAXPRINT 20
00087
00090 #define NOSH_MAXPOP 20
00091
00096 enum eNOsh_MolFormat {
00097     NMF_PQR=0,
00098     NMF_PDB=1,
00099     NMF_XML=2
00100 };
00101
00106 typedef enum eNOsh_MolFormat Nosh_MolFormat;
00107
00112 enum eNOsh_CalcType {
00113     NCT_MG=0,
00114     NCT_FEM=1,
00115     NCT_APOL=2
00116 };
00117
00122 typedef enum eNOsh_CalcType Nosh_CalcType;
00123
00128 enum eNOsh_ParmFormat {
00129     NPF_FLAT=0,
00130     NPF_XML=1
00131 };
00132
00137 typedef enum eNOsh_ParmFormat Nosh_ParmFormat;
00138
00143 enum eNOsh_PrintType {
00144     NPT_ENERGY=0,
00145     NPT_FORCE=1,
00146     NPT_ELECENERGY,
00147     NPT_ELECFORCE,
00148     NPT_APOLENERGY,
00149     NPT_APOLFORCE
```

```

00150 };
00151
00156 typedef enum eNosh_PrintType Nosh_PrintType;
00157
00163 struct sNosh_calc {
00164     MGparm *mgparm;
00165     FEMparm *femparm;
00166     PBEparm *pbeparm;
00167     APOLparm *apolparm;
00168     Nosh_CalcType calctype;
00169 };
00170
00175 typedef struct sNosh_calc Nosh_calc;
00176
00182 struct sNosh {
00183
00184     Nosh_calc *calc[NOSH_MAXCALC];
00185         int ncalc;
00186     Nosh_calc *elec[NOSH_MAXCALC];
00187         int nelec;
00188     Nosh_calc *apol[NOSH_MAXCALC];
00189         int napol;
00190         int ispara;
00191         int proc_rank;
00192         int proc_size;
00193         int bogus;
00194         int elec2calc[NOSH_MAXCALC];
00195     int apol2calc[NOSH_MAXCALC];
00196         int nmol;
00197         char molpath[NOSH_MAXMOL] [VMAX_ARGLEN];
00198         Nosh_MolFormat molfmt[NOSH_MAXMOL];
00199     Valist *alist[NOSH_MAXMOL];
00200         int gotparm;
00201         char parmpath[VMAX_ARGLEN];
00202         Nosh_ParmFormat parmfmt;
00203         int ndiel;
00204         char dielXpath[NOSH_MAXMOL] [VMAX_ARGLEN];
00205         char dielYpath[NOSH_MAXMOL] [VMAX_ARGLEN];
00206         char dielZpath[NOSH_MAXMOL] [VMAX_ARGLEN];
00207         Vdata_Format dielfmt[NOSH_MAXMOL];
00208         int nkappa;
00209         char kappapath[NOSH_MAXMOL] [VMAX_ARGLEN];
00210         Vdata_Format kappafmt[NOSH_MAXMOL];
00211     int npot;
00212         char potpath[NOSH_MAXMOL] [VMAX_ARGLEN];
00213         Vdata_Format potfmt[NOSH_MAXMOL];
00214         int ncharge;
00215         char chargepath[NOSH_MAXMOL] [VMAX_ARGLEN];
00216         Vdata_Format chargefmt[NOSH_MAXMOL];
00217         int nmesh;
00218         char meshpath[NOSH_MAXMOL] [VMAX_ARGLEN];
00219         Vdata_Format meshfmt[NOSH_MAXMOL];
00220         int nprint;
00221         Nosh_PrintType printwhat[NOSH_MAXPRINT];
00222         int printnarg[NOSH_MAXPRINT];
00223         int printcalc[NOSH_MAXPRINT] [NOSH_MAXPOP];
00224         int printop[NOSH_MAXPRINT] [NOSH_MAXPOP];

```

```

00253     int parsed;
00254     char elecname[NOSH_MAXCALC] [VMAX_ARGLEN];
00255     char apolname[NOSH_MAXCALC] [VMAX_ARGLEN];
00258 };
00259
00264 typedef struct sNOsh NOsh;
00265
00266 /* //////////////////////////////// */
00267 // Class NOsh: Inlineable methods (mcsh.c)
00269 #if !defined(VINLINE_NOSH)
00277 VEXTERNC char* NOsh_getMolpath(NOsh *thee, int imol);
00278
00286 VEXTERNC char* NOsh_getDielXpath(NOsh *thee, int imap);
00287
00295 VEXTERNC char* NOsh_getDielYpath(NOsh *thee, int imap);
00296
00304 VEXTERNC char* NOsh_getDielZpath(NOsh *thee, int imap);
00305
00313 VEXTERNC char* NOsh_getKappapath(NOsh *thee, int imap);
00314
00322 VEXTERNC char* NOsh_getPotpath(NOsh *thee, int imap);
00323
00331 VEXTERNC char* NOsh_getChargepath(NOsh *thee, int imap);
00332
00340 VEXTERNC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc);
00341
00349 VEXTERNC int NOsh_getDielfmt(NOsh *thee, int imap);
00350
00358 VEXTERNC int NOsh_getKappafmt(NOsh *thee, int imap);
00359
00367 VEXTERNC int NOsh_getPotfmt(NOsh *thee, int imap);
00368
00376 VEXTERNC int NOsh_getChargefmt(NOsh *thee, int imap);
00377
00378 #else
00379
00380 # define NOsh_getMolpath(thee, imol) ((thee)->molpath[(imol)])
00381 # define NOsh_getDielXpath(thee, imol) ((thee)->dielXpath[(imol)])
00382 # define NOsh_getDielYpath(thee, imol) ((thee)->dielYpath[(imol)])
00383 # define NOsh_getDielZpath(thee, imol) ((thee)->dielZpath[(imol]))
00384 # define NOsh_getKappapath(thee, imol) ((thee)->kappapath[(imol)])
00385 # define NOsh_getPotpath(thee, imol) ((thee)->potpath[(imol]))
00386 # define NOsh_getChargepath(thee, imol) ((thee)->chargepath[(imol]))
00387 # define NOsh_getCalc(thee, icalc) ((thee)->calc[(icalc)])
00388 # define NOsh_getDielfmt(thee, imap) ((thee)->dielfmt[(imap)])
00389 # define NOsh_getKappafmt(thee, imap) ((thee)->kappafmt[(imap)])
00390 # define NOsh_getPotfmt(thee, imap) ((thee)->potfmt[(imap]))
00391 # define NOsh_getChargefmt(thee, imap) ((thee)->chargefmt[(imap]))
00392
00393 #endif
00394
00395
00396 /* //////////////////////////////// */
00397 // Class NOsh: Non-inlineable methods (mcsh.c)
00399
00407 VEXTERNC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint);
00408

```

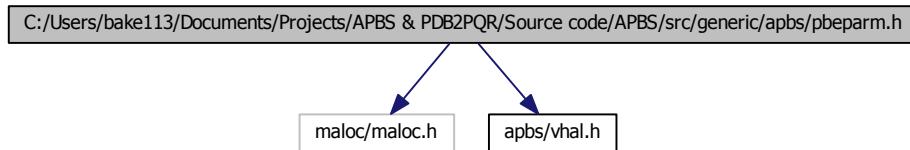
```
00418 VEXTERNC char* NOsh_elecname(NOsh *thee, int ielec);
00419
00420 VEXTERNC int NOsh_elec2calc(NOsh *thee, int icalc);
00421
00422 VEXTERNC int NOsh_apol2calc(NOsh *thee, int icalc);
00423
00424 VEXTERNC int NOsh_printNarg(NOsh *thee, int iprint);
00425
00426 VEXTERNC int NOsh_printOp(NOsh *thee, int iprint, int iarg);
00427
00428 VEXTERNC int NOsh_printCalc(NOsh *thee, int iprint, int iarg);
00429
00430 VEXTERNC NOsh* NOsh_ctor(int rank, int size);
00431
00432 VEXTERNC NOsh_calc* NOsh_calc_ctor(
00433     NOsh_CalcType calcType
00434 );
00435
00436 VEXTERNC int NOsh_calc_copy(
00437     NOsh_calc *thee,
00438     NOsh_calc *source
00439 );
00440
00441 VEXTERNC void NOsh_calc_dtor(NOsh_calc **thee);
00442
00443 VEXTERNC int NOsh_ctor2(NOsh *thee, int rank, int size);
00444
00445 VEXTERNC void NOsh_dtor(NOsh **thee);
00446
00447 VEXTERNC void NOsh_dtor2(NOsh *thee);
00448
00449 VEXTERNC int NOsh_parseInput(NOsh *thee, Vio *sock);
00450
00451 VEXTERNC int NOsh_parseInputFile(NOsh *thee, char *filename);
00452
00453 VEXTERNC int NOsh_setupElecCalc(
00454     NOsh *thee,
00455     Valist *alist[NOSH_MAXMOL]
00456 );
00457
00458 VEXTERNC int NOsh_setupApolCalc(
00459     NOsh *thee,
00460     Valist *alist[NOSH_MAXMOL]
00461 );
00462
00463 #endif
00464
```

**10.27 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source
code/APBS/src/generic/apbs/pbeparm.h File Reference**

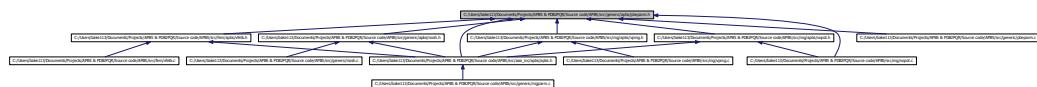
Contains declarations for class PBParm.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"

Include dependency graph for pbeparm.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sPBEParm`
Parameter structure for PBE variables from input files.

Defines

- #define `PBEPARM_MAXWRITE` 20
Number of things that can be written out in a single calculation.

Typedefs

- typedef enum `ePBEParm_calcEnergy` `PBEParm_calcEnergy`
Define `ePBEParm_calcEnergy` enumeration as `PBEParm_calcEnergy`.
- typedef enum `ePBEParm_calcForce` `PBEParm_calcForce`
Define `ePBEParm_calcForce` enumeration as `PBEParm_calcForce`.
- typedef struct `sPBEParm` `PBEParm`
Declaration of the `PBEParm` class as the `PBEParm` structure.

Enumerations

- enum `ePBParm_calcEnergy` { `PCE_NO` = 0, `PCE_TOTAL` = 1, `PCE_COMPS` = 2 }
Define energy calculation enumeration.
- enum `ePBParm_calcForce` { `PCF_NO` = 0, `PCF_TOTAL` = 1, `PCF_COMPS` = 2 }
Define force calculation enumeration.

Functions

- VEXTERNC double `PBParm_getIonCharge` (`PBParm` *`thee`, int `iion`)
Get charge (e) of specified ion species.
- VEXTERNC double `PBParm_getIonConc` (`PBParm` *`thee`, int `iion`)
Get concentration (M) of specified ion species.
- VEXTERNC double `PBParm_getIonRadius` (`PBParm` *`thee`, int `iion`)
Get radius (A) of specified ion species.
- VEXTERNC `PBParm` * `PBParm_ctor` ()
Construct PBParm object.
- VEXTERNC int `PBParm_ctor2` (`PBParm` *`thee`)
FORTRAN stub to construct PBParm object.
- VEXTERNC void `PBParm_dtor` (`PBParm` **`thee`)
Object destructor.
- VEXTERNC void `PBParm_dtor2` (`PBParm` *`thee`)
FORTRAN stub for object destructor.
- VEXTERNC int `PBParm_check` (`PBParm` *`thee`)
Consistency check for parameter values stored in object.
- VEXTERNC void `PBParm_copy` (`PBParm` *`thee`, `PBParm` *`parm`)
Copy PBParm object into thee.
- VEXTERNC int `PBParm_parseToken` (`PBParm` *`thee`, char `tok`[`VMAX_BUFSIZE`],
`Vio` *`sock`)
Parse a keyword from an input file.

10.27.1 Detailed Description

Contains declarations for class `PBParm`.

Version

Id:

[pbeparm.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [pbeparm.h](#).

10.28 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/pbeparm.h

```
00001
00062 #ifndef _PBEPARM_H_
00063 #define _PBEPARM_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067
00068 /* Headers specific to this file */
00069 #include "apbs/vhal.h"
00070
00074 #define PBEPARM_MAXWRITE 20
00075
00080 enum ePBEParm_calcEnergy {
00081     PCE_NO=0,
00082     PCE_TOTAL=1,
00083     PCE_COMPS=2
00084 };
00085
00090 typedef enum ePBEParm_calcEnergy PBEParm_calcEnergy;
00091
00096 enum ePBEParm_calcForce {
00097     PCF_NO=0,
00098     PCF_TOTAL=1,
00099     PCF_COMPS=2
00100 };
00101
00106 typedef enum ePBEParm_calcForce PBEParm_calcForce;
00107
00116 struct sPBEParm {
00117
00118     int molid;
00119     int setmolid;
00120     int useDielMap;
00122     int dielMapID;
00123     int useKappaMap;
00125     int kappaMapID;
00126     int usePotMap;
00128     int potMapID;
00130     int useChargeMap;
00132     int chargeMapID;
00133     Vhal_PBEType pbetype;
00134     int setpbetype;
00135     Vbcfl bcfl;
00136     int setbcfl;
00137     int nion;
00138     int setnion;
00139     double ionq[MAXION];
00140     double ionc[MAXION];
00141     double ionr[MAXION];
00142     int setion[MAXION];
00143     double pdie;
00144     int setpdie;
00145     double sdens;
```

```
00146     int setsdens;
00147     double sdie;
00148     int setsdie;
00149     Vsurf_Meth srfm;
00150     int setsrfm;
00151     double srad;
00152     int setsrad;
00153     double swin;
00154     int setswin;
00155     double temp;
00156     int settemp;
00158     double smsize;
00159     int setsmsize;
00161     double smvolume;
00162     int setsmvolume;
00164     PBEparm_calcEnergy calcenergy;
00165     int setcalcenergy;
00166     PBEparm_calcForce calcforce;
00167     int setcalcforce;
00169 /*-----*/
00170 /* Added by Michael Grabe */
00171 /*-----*/
00172
00173     double zmem;
00174     int setzmem;
00175     double Lmem;
00176     int setLmem;
00177     double mdie;
00178     int setmdie;
00179     double memv;
00180     int setmemv;
00182 /*-----*/
00183
00184     int numwrite;
00185     char writestem[PBEPARM_MAXWRITE] [VMAX_ARGLEN];
00187     Vdata_Type writetype[PBEPARM_MAXWRITE];
00188     Vdata_Format writefmt[PBEPARM_MAXWRITE];
00189     int writemat;
00193     int setwritemat;
00194     char writematstem[VMAX_ARGLEN];
00195     int writematflag;
00200     int parsed;
00202 };
00203
00208 typedef struct sPBEparm PBEparm;
00209
00210 /* /////////////////////////////////
00211 // Class NOsh: Non-inlineable methods (mcsh.c)
00213
00219 VEXTERNC double PBEparm_getIonCharge(
00220     PBEparm *thee,
00221     int iion
00222 );
00223
00229 VEXTERNC double PBEparm_getIonConc(
00230     PBEparm *thee,
00231     int iion
```

```
00232     );
00233
00239 VEXTERNC double PBparm_getIonRadius(
00240     PBparm *thee,
00241     int iion
00242 );
00243
00244
00250 VEXTERNC PBparm* PBparm_ctor();
00251
00257 VEXTERNC int PBparm_ctor2(
00258     PBparm *thee
00259 );
00260
00265 VEXTERNC void PBparm_dtor(
00266     PBparm **thee
00267 );
00268
00273 VEXTERNC void PBparm_dtor2(
00274     PBparm *thee
00275 );
00276
00282 VEXTERNC int PBparm_check(
00283     PBparm *thee
00284 );
00285
00290 VEXTERNC void PBparm_copy(
00291     PBparm *thee,
00292     PBparm *parm
00293 );
00294
00301 VEXTERNC int PBparm_parseToken(
00302     PBparm *thee,
00303     char tok[VMAX_BUFSIZE],
00304     Vio *sock
00305 );
00306
00307
00308 #endif
00309
```

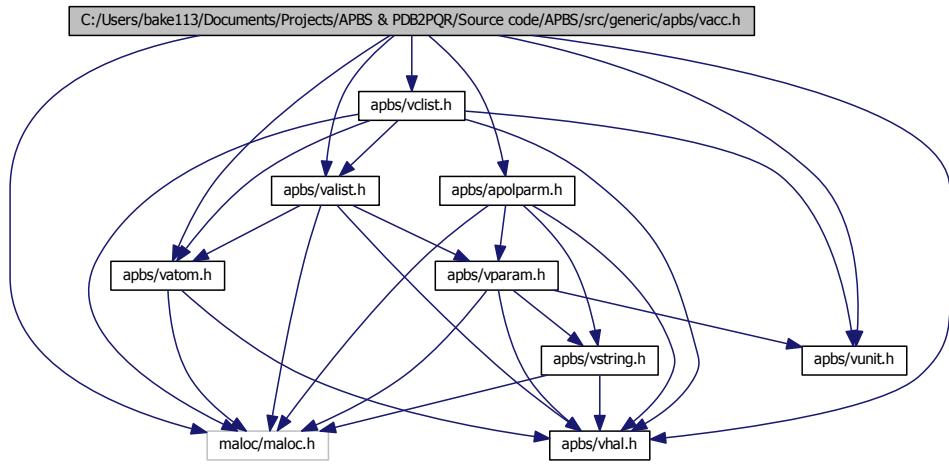
10.29 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vacc.h File Reference

Contains declarations for class Vacc.

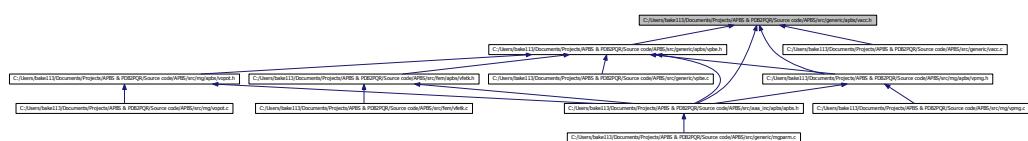
```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/valist.h"
#include "apbs/vclist.h"
```

```
#include "apbs/vatom.h"
#include "apbs/vunit.h"
#include "apbs/apolparm.h"

Include dependency graph for vacc.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sVaccSurf`
Surface object list of per-atom surface points.
- struct `sVacc`
Oracle for solvent- and ion-accessibility around a biomolecule.

Typedefs

- **typedef struct sVaccSurf VaccSurf**
Declaration of the VaccSurf class as the VaccSurf structure.
- **typedef struct sVacc Vacc**
Declaration of the Vacc class as the Vacc structure.

Functions

- **VEXTERNC unsigned long int Vacc_memChk (Vacc *thee)**
Get number of bytes in this object and its members.
- **VEXTERNC VaccSurf * VaccSurf_ctor (Vmem *mem, double probe_radius, int nsphere)**
Allocate and construct the surface object; do not assign surface points to positions.
- **VEXTERNC int VaccSurf_ctor2 (VaccSurf *thee, Vmem *mem, double probe_radius, int nsphere)**
Construct the surface object using previously allocated memory; do not assign surface points to positions.
- **VEXTERNC void VaccSurf_dtor (VaccSurf **thee)**
Destroy the surface object and free its memory.
- **VEXTERNC void VaccSurf_dtor2 (VaccSurf *thee)**
Destroy the surface object.
- **VEXTERNC VaccSurf * VaccSurf_refSphere (Vmem *mem, int npts)**
Set up an array of points for a reference sphere of unit radius.
- **VEXTERNC VaccSurf * Vacc_atomSurf (Vacc *thee, Vatom *atom, VaccSurf *ref, double probe_radius)**
Set up an array of points corresponding to the SAS due to a particular atom.
- **VEXTERNC Vacc * Vacc_ctor (Valist *alist, Vclist *clist, double surf_density)**
Construct the accessibility object.
- **VEXTERNC int Vacc_ctor2 (Vacc *thee, Valist *alist, Vclist *clist, double surf_density)**
FORTRAN stub to construct the accessibility object.
- **VEXTERNC void Vacc_dtor (Vacc **thee)**
Destroy object.
- **VEXTERNC void Vacc_dtor2 (Vacc *thee)**
FORTRAN stub to destroy object.
- **VEXTERNC double Vacc_vdwAcc (Vacc *thee, double center[VAPBS_DIM])**
Report van der Waals accessibility.
- **VEXTERNC double Vacc_ivdwAcc (Vacc *thee, double center[VAPBS_DIM], double radius)**

Report inflated van der Waals accessibility.

- VEXTERNC double `Vacc_molAcc` (`Vacc *thee, double center[VAPBS_DIM], double radius`)

Report molecular accessibility.

- VEXTERNC double `Vacc_fastMolAcc` (`Vacc *thee, double center[VAPBS_DIM], double radius`)

Report molecular accessibility quickly.

- VEXTERNC double `Vacc_splineAcc` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad`)

Report spline-based accessibility.

- VEXTERNC void `Vacc_splineAccGrad` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, double *grad`)

Report gradient of spline-based accessibility.

- VEXTERNC double `Vacc_splineAccAtom` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom`)

Report spline-based accessibility for a given atom.

- VEXTERNC void `Vacc_splineAccGradAtomUnnorm` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom, double *force`)

Report gradient of spline-based accessibility with respect to a particular atom (see `Vpmg_splineAccAtom`)

- VEXTERNC void `Vacc_splineAccGradAtomNorm` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom, double *force`)

Report gradient of spline-based accessibility with respect to a particular atom normalized by the accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)

- VEXTERNC void `Vacc_splineAccGradAtomNorm4` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom, double *force`)

Report gradient of spline-based accessibility with respect to a particular atom normalized by a 4th order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)

- VEXTERNC void `Vacc_splineAccGradAtomNorm3` (`Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom, double *force`)

Report gradient of spline-based accessibility with respect to a particular atom normalized by a 3rd order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)

- VEXTERNC double `Vacc_SASA` (`Vacc *thee, double radius`)

Build the solvent accessible surface (SAS) and calculate the solvent accessible surface area.

- VEXTERNC double `Vacc_totalSASA` (`Vacc *thee, double radius`)

Return the total solvent accessible surface area (SASA)

- VEXTERNC double `Vacc_atomSASA` (`Vacc *thee, double radius, Vatom *atom`)

Return the atomic solvent accessible surface area (SASA)

- VEXTERNC `VaccSurf * Vacc_atomSASPoints (Vacc *thee, double radius, Vatom *atom)`

Get the set of points for this atom's solvent-accessible surface.
- VEXTERNC void `Vacc_atomdSAV (Vacc *thee, double radius, Vatom *atom, double *dSA)`

Get the derivative of solvent accessible volume.
- VEXTERNC void `Vacc_atomdSASA (Vacc *thee, double dpos, double radius, Vatom *atom, double *dSA)`

Get the derivative of solvent accessible area.
- VEXTERNC void `Vacc_totalAtomdSASA (Vacc *thee, double dpos, double radius, Vatom *atom, double *dSA)`

Testing purposes only.
- VEXTERNC void `Vacc_totalAtomdSAV (Vacc *thee, double dpos, double radius, Vatom *atom, double *dSA, Vclist *clist)`

Total solvent accessible volume.
- VEXTERNC double `Vacc_totalsAV (Vacc *thee, Vclist *clist, APOLparm *apolparm, double radius)`

Return the total solvent accessible volume (SAV)
- VEXTERNC int `Vacc_wcaEnergy (Vacc *thee, APOLparm *apolparm, Valist *alist, Vclist *clist)`

Return the WCA integral energy.
- VEXTERNC int `Vacc_wcaForceAtom (Vacc *thee, APOLparm *apolparm, Vclist *clist, Vatom *atom, double *force)`

Return the WCA integral force.
- VEXTERNC int `Vacc_wcaEnergyAtom (Vacc *thee, APOLparm *apolparm, Valist *alist, Vclist *clist, int iatom, double *value)`

Calculate the WCA energy for an atom.

10.29.1 Detailed Description

Contains declarations for class Vacc.

Version

Id:

`vacc.h 1667 2011-12-02 23:22:02Z pcellis`

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*
```

Definition in file [vacc.h](#).

10.30 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vacc.h

```
00001
00062 #ifndef _VACC_H_
00063 #define _VACC_H_
00064
00065 /* Generic headers */
00066 #include "maloc/malloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Headers specific to this file */
00070 #include "apbs/valist.h"
00071 #include "apbs/vclist.h"
00072 #include "apbs/vatom.h"
00073 #include "apbs/vunit.h"
00074 #include "apbs/apolparm.h"
00075
00081 struct sVaccSurf {
00082     Vmem *mem;
00083     double *xpts;
00084     double *ypts;
00085     double *zpts;
00086     char *bpts;
00088     double area;
00089     int npts;
00090     double probe_radius;
00092 };
00093
00098 typedef struct sVaccSurf VaccSurf;
00099
00105 struct sVacc {
00106
00107     Vmem *mem;
00108     Valist *alist;
00109     Vclist *clist;
00110     int *atomFlags;
00113     VaccSurf *refSphere;
00114     VaccSurf **surf;
00117     Vset acc;
00119     double surf_density;
00122 };
00123
00128 typedef struct sVacc Vacc;
00129
00130 #if !defined(VINLINE_VACC)
00131
00137     VEXTERNC unsigned long int Vacc_memChk(
00138             Vacc *thee
00139         );
00140
00141 #else /* if defined(VINLINE_VACC) */
00142
00143 #    define Vacc_memChk(thee) (Vmem_bytes((thee)->mem))
00144
00145 #endif /* if !defined(VINLINE_VACC) */
```

```
00146
00154 VEXTERNC VaccSurf* VaccSurf_ctor(
00155     Vmem *mem,
00156     double probe_radius,
00157     int nsphere
00158 );
00159
00167 VEXTERNC int VaccSurf_ctor2(
00168     VaccSurf *thee,
00169     Vmem *mem,
00170     double probe_radius,
00171     int nsphere
00172 );
00173
00179 VEXTERNC void VaccSurf_dtor(
00180     VaccSurf **thee
00181 );
00182
00188 VEXTERNC void VaccSurf_dtor2(
00189     VaccSurf *thee
00190 );
00191
00206 VEXTERNC VaccSurf* VaccSurf_refSphere(
00207     Vmem *mem,
00208     int npts
00209 );
00210
00218 VEXTERNC VaccSurf* Vacc_atomSurf(
00219     Vacc *thee,
00220     Vatom *atom,
00221     VaccSurf *ref,
00223     double probe_radius
00224 );
00225
00226
00231 VEXTERNC Vacc* Vacc_ctor(
00232     Valist *alist,
00233     Vclist *clist,
00235     double surf_density
00237 );
00238
00243 VEXTERNC int Vacc_ctor2(
00244     Vacc *thee,
00245     Valist *alist,
00246     Vclist *clist,
00248     double surf_density
00250 );
00251
00256 VEXTERNC void Vacc_dtor(
00257     Vacc **thee
00258 );
00259
00264 VEXTERNC void Vacc_dtor2(
00265     Vacc *thee
00266 );
00267
00278 VEXTERNC double Vacc_vdwAcc(
```

```

00279     Vacc *thee,
00280     double center[VAPBS_DIM]
00281 );
00282
00294 VEXTERNC double Vacc_ivdwAcc(
00295     Vacc *thee,
00296     double center[VAPBS_DIM],
00297     double radius
00298 );
00299
00314 VEXTERNC double Vacc_molAcc(
00315     Vacc *thee,
00316     double center[VAPBS_DIM],
00317     double radius
00318 );
00319
00338 VEXTERNC double Vacc_fastMolAcc(
00339     Vacc *thee,
00340     double center[VAPBS_DIM],
00341     double radius
00342 );
00343
00355 VEXTERNC double Vacc_splineAcc(
00356     Vacc *thee,
00357     double center[VAPBS_DIM],
00358     double win,
00359     double inftrad
00360 );
00361
00367 VEXTERNC void Vacc_splineAccGrad(
00368     Vacc *thee,
00369     double center[VAPBS_DIM],
00370     double win,
00371     double inftrad,
00372     double *grad
00373 );
00374
00386 VEXTERNC double Vacc_splineAccAtom(
00387     Vacc *thee,
00388     double center[VAPBS_DIM],
00389     double win,
00390     double inftrad,
00391     Vatom *atom
00392 );
00393
00404 VEXTERNC void Vacc_splineAccGradAtomUnnorm(
00405     Vacc *thee,
00406     double center[VAPBS_DIM],
00407     double win,
00408     double inftrad,
00409     Vatom *atom,
00410     double *force
00411 );
00412
00424 VEXTERNC void Vacc_splineAccGradAtomNorm(
00425     Vacc *thee,
00426     double center[VAPBS_DIM],

```

```
00427     double win,
00428     double infrad,
00429     Vatom *atom,
00430     double *force
00431   );
00432
00440 VEXTERNC void Vacc_splineAccGradAtomNorm4(
00441     Vacc *thee,
00442     double center[VAPBS_DIM],
00443     double win,
00444     double infrad,
00445     Vatom *atom,
00446     double *force
00447   );
00448
00456 VEXTERNC void Vacc_splineAccGradAtomNorm3(
00457     Vacc *thee,
00458     double center[VAPBS_DIM],
00459     double win,
00460     double infrad,
00461     Vatom *atom,
00462     double *force
00463   );
00464
00465
00475 VEXTERNC double Vacc_SASA(
00476     Vacc *thee,
00477     double radius
00478   );
00479
00487 VEXTERNC double Vacc_totalSASA(
00488     Vacc *thee,
00489     double radius
00490   );
00491
00499 VEXTERNC double Vacc_atomSASA(
00500     Vacc *thee,
00501     double radius,
00502     Vatom *atom
00503   );
00504
00511 VEXTERNC VaccSurf* Vacc_atomSASPoints(
00512     Vacc *thee,
00513     double radius,
00514     Vatom *atom
00515   );
00516
00522 VEXTERNC void Vacc_atomdSAV(
00523     Vacc *thee,
00524     double radius,
00525     Vatom *atom,
00526     double *dSA
00527   );
00528
00534 VEXTERNC void Vacc_atomdSASA(
00535     Vacc *thee,
00536     double dpos,
```

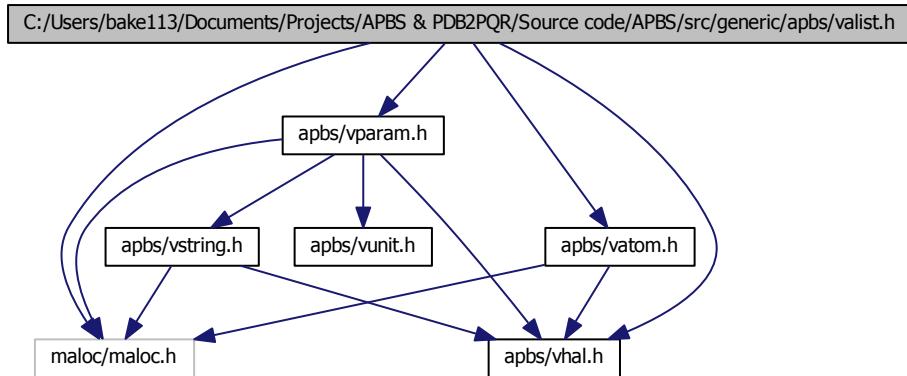
```
00537     double radius,
00538     Vatom *atom,
00539     double *dSA
00540 );
00541
00542 VEXTERNC void Vacc_totalAtomdSASA(
00543     Vacc *thee,
00544     double dpos,
00545     double radius,
00546     Vatom *atom,
00547     double *dSA
00548 );
00549
00550 VEXTERNC void Vacc_totalAtomdSAV(
00551     Vacc *thee,
00552     double dpos,
00553     double radius,
00554     Vatom *atom,
00555     double *dSA,
00556     Vclist *clist
00557 );
00558
00559 VEXTERNC double Vacc_totalsAV(
00560     Vacc *thee,
00561     Vclist *clist,
00562     APOLparm *apolparm,
00563     double radius
00564 );
00565
00566 VEXTERNC int Vacc_wcaEnergy(
00567     Vacc *thee,
00568     APOLparm *apolparm,
00569     Valist *alist,
00570     Vclist *clist
00571 );
00572
00573 VEXTERNC int Vacc_wcaForceAtom(Vacc *thee,
00574     APOLparm *apolparm,
00575     Vclist *clist,
00576     Vatom *atom,
00577     double *force
00578 );
00579
00580 VEXTERNC int Vacc_wcaEnergyAtom(
00581     Vacc *thee,
00582     APOLparm *apolparm,
00583     Valist *alist,
00584     Vclist *clist,
00585     int iatom,
00586     double *value
00587 );
00588
00589 #endif /* ifndef _VACC_H_ */
```

10.31 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/valist.h File Reference

Contains declarations for class Valist.

```
#include "maloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vatom.h"
#include "apbs/vparam.h"
```

Include dependency graph for valist.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sValist](#)

Container class for list of atom objects.

Typedefs

- **typedef struct sValist Valist**

Declaration of the Valist class as the Valist structure.

Functions

- **VEXTERNC Vatom * Valist_getAtomList (Valist *thee)**

Get actual array of atom objects from the list.

- **VEXTERNC double Valist_getCenterX (Valist *thee)**

Get x-coordinate of molecule center.

- **VEXTERNC double Valist_getCenterY (Valist *thee)**

Get y-coordinate of molecule center.

- **VEXTERNC double Valist_getCenterZ (Valist *thee)**

Get z-coordinate of molecule center.

- **VEXTERNC int Valist_getNumberAtoms (Valist *thee)**

Get number of atoms in the list.

- **VEXTERNC Vatom * Valist_getAtom (Valist *thee, int i)**

Get pointer to particular atom in list.

- **VEXTERNC unsigned long int Valist_memChk (Valist *thee)**

Get total memory allocated for this object and its members.

- **VEXTERNC Valist * Valist_ctor ()**

Construct the atom list object.

- **VEXTERNC Vrc_Codes Valist_ctor2 (Valist *thee)**

FORTRAN stub to construct the atom list object.

- **VEXTERNC void Valist_dtor (Valist **thee)**

Destroys atom list object.

- **VEXTERNC void Valist_dtor2 (Valist *thee)**

FORTRAN stub to destroy atom list object.

- **VEXTERNC Vrc_Codes Valist_readPQR (Valist *thee, Vparam *param, Vio *sock)**

Fill atom list with information from a PQR file.

- **VEXTERNC Vrc_Codes Valist_readPDB (Valist *thee, Vparam *param, Vio *sock)**

Fill atom list with information from a PDB file.

- **VEXTERNC Vrc_Codes Valist_readXML (Valist *thee, Vparam *param, Vio *sock)**

Fill atom list with information from an XML file.

- VEXTERNC Vrc_Codes [Valist_getStatistics](#) ([Valist](#) *thee)

Load up Valist with various statistics.

10.31.1 Detailed Description

Contains declarations for class Valist.

Version

Id:

[valist.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without
```

```
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file valist.h.

10.32 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/valist.h

```
00001  
00062 #ifndef _VALIST_H_  
00063 #define _VALIST_H_  
00064  
00065 /* Generic headers */  
00066 #include "maloc/maloc.h"  
00067 #include "apbs/vhal.h"  
00068  
00069 /* Headers specific to this file */  
00070 #include "apbs/vatom.h"  
00071 #include "apbs/vparam.h"  
00072  
00078 struct sValist {  
00079  
00080     int number;  
00081     double center[3];  
00082     double mincrd[3];  
00083     double maxcrd[3];  
00084     double maxrad;  
00085     double charge;  
00086     Vatom *atoms;  
00087     Vmem *vmem;  
00089 };  
00090  
00095 typedef struct sValist Valist;  
00096  
00097 #if !defined(VINLINE_VATOM)  
00098  
00105 VEXTERNC Vatom* Valist_getAtomList(  
00106         Valist *thee  
00107     );  
00108
```

```
00114 VEXTERNC double Valist_getCenterX(
00115     Valist *thee
00116 );
00117
00123 VEXTERNC double Valist_getCenterY(
00124     Valist *thee
00125 );
00126
00132 VEXTERNC double Valist_getCenterZ(
00133     Valist *thee
00134 );
00135
00141 VEXTERNC int Valist_getNumberAtoms(
00142     Valist *thee
00143 );
00144
00150 VEXTERNC Vatom* Valist_getAtom(
00151     Valist *thee,
00152     int i
00153 );
00154
00160 VEXTERNC unsigned long int Valist_memChk(
00161     Valist *thee
00162 );
00163
00164 #else /* if defined(VINLINE_VATOM) */
00165 #    define Valist_getAtomList(thee) ((thee)->atoms)
00166 #    define Valist_getNumberAtoms(thee) ((thee)->number)
00167 #    define Valist_getAtom(thee, i) (&((thee)->atoms[i]))
00168 #    define Valist_memChk(thee) (Vmem_bytes((thee)->vmem))
00169 #    define Valist_getCenterX(thee) ((thee)->center[0])
00170 #    define Valist_getCenterY(thee) ((thee)->center[1])
00171 #    define Valist_getCenterZ(thee) ((thee)->center[2])
00172 #endif /* if !defined(VINLINE_VATOM) */
00173
00179 VEXTERNC Valist* Valist_ctor();
00180
00186 VEXTERNC Vrc_Codes Valist_ctor2(
00187     Valist *thee
00188 );
00189
00194 VEXTERNC void Valist_dtor(
00195     Valist **thee
00196 );
00197
00202 VEXTERNC void Valist_dtor2(
00203     Valist *thee
00204 );
00205
00217 VEXTERNC Vrc_Codes Valist_readPQR(
00218     Valist *thee,
00219     Vparam *param,
00220     Vio *sock
00221 );
00222
00232 VEXTERNC Vrc_Codes Valist_readPDB(
00233     Valist *thee,
```

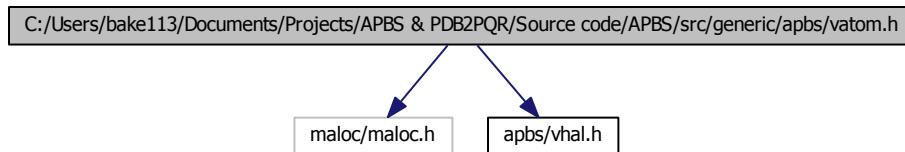
```
00234     Vparam *param,
00235     Vio *sock
00236 );
00237
00247 VEXTERNC Vrc_Codes Valist_readXML(
00248     Valist *thee,
00249     Vparam *param,
00250     Vio *sock
00251 );
00252
00259 VEXTERNC Vrc_Codes Valist_getStatistics(Valist *thee);
00260
00261
00262 #endif /* ifndef _VALIST_H_ */
```

10.33 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vatom.h File Reference

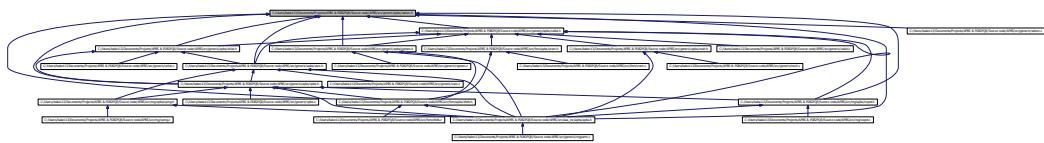
Contains declarations for class Vatom.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
```

Include dependency graph for vatom.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **sVatom**

Contains public data members for Vatom class/module.

Defines

- #define **VMAX_RECLEN** 64

Residue name length.

Typedefs

- typedef struct **sVatom** **Vatom**

Declaration of the Vatom class as the Vatom structure.

Functions

- VEXTERNC double * **Vatom_getPosition** (**Vatom** *thee)
Get atomic position.
- VEXTERNC void **Vatom_setRadius** (**Vatom** *thee, double radius)
Set atomic radius.
- VEXTERNC double **Vatom_getRadius** (**Vatom** *thee)
Get atomic position.
- VEXTERNC void **Vatom_setPartID** (**Vatom** *thee, int partID)
Set partition ID.
- VEXTERNC double **Vatom_getPartID** (**Vatom** *thee)
Get partition ID.
- VEXTERNC void **Vatom_setAtomID** (**Vatom** *thee, int id)
Set atom ID.
- VEXTERNC double **Vatom_getAtomID** (**Vatom** *thee)
Get atom ID.
- VEXTERNC void **Vatom_setCharge** (**Vatom** *thee, double charge)
Set atomic charge.
- VEXTERNC double **Vatom_getCharge** (**Vatom** *thee)
Get atomic charge.
- VEXTERNC void **Vatom_setEpsilon** (**Vatom** *thee, double epsilon)
Set atomic epsilon.
- VEXTERNC double **Vatom_getEpsilon** (**Vatom** *thee)
Get atomic epsilon.

- VEXTERNC unsigned long int `Vatom_memChk` (`Vatom *thee`)
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC void `Vatom_setResName` (`Vatom *thee, char resName[VMAX_RECLEN]`)

Set residue name.
- VEXTERNC void `Vatom_setAtomName` (`Vatom *thee, char atomName[VMAX_RECLEN]`)

Set atom name.
- VEXTERNC void `Vatom_getResName` (`Vatom *thee, char resName[VMAX_RECLEN]`)

Retrieve residue name.
- VEXTERNC void `Vatom_getAtomName` (`Vatom *thee, char atomName[VMAX_RECLEN]`)

Retrieve atom name.
- VEXTERNC `Vatom * Vatom_ctor` ()

Constructor for the Vatom class.
- VEXTERNC int `Vatom_ctor2` (`Vatom *thee`)

FORTRAN stub constructor for the Vatom class.
- VEXTERNC void `Vatom_dtor` (`Vatom **thee`)

Object destructor.
- VEXTERNC void `Vatom_dtor2` (`Vatom *thee`)

FORTRAN stub object destructor.
- VEXTERNC void `Vatom_setPosition` (`Vatom *thee, double position[3]`)

Set the atomic position.
- VEXTERNC void `Vatom_copyTo` (`Vatom *thee, Vatom *dest`)

Copy information to another atom.
- VEXTERNC void `Vatom_copyFrom` (`Vatom *thee, Vatom *src`)

Copy information to another atom.

10.33.1 Detailed Description

Contains declarations for class Vatom.

Version

Id:

`vatom.h` 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*
```

Definition in file [vatom.h](#).

10.34 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vatom.h

```
00001
00062 #ifndef _VATOM_H_
00063 #define _VATOM_H_
00064
00065 #include "maloc/maloc.h"
00066 #include "apbs/vhal.h"
00067
00074 #define VMAX_RECLEN      64
00075
00081 struct sVatom {
00082
00083     double position[3];
00084     double radius;
00085     double charge;
00086     double partID;
00088     double epsilon;
00090     int id;
00094     char resName[VMAX_RECLEN];
00095     char atomName[VMAX_RECLEN];
00097 #if defined(WITH_TINKER)
00098
00099     double dipole[3];
00100     double quadrupole[9];
00101     double inducedDipole[3];
00102     double nlInducedDipole[3];
00104 #endif /* if defined(WITH_TINKER) */
00105 };
00106
00111 typedef struct sVatom Vatom;
00112
00113 #if !defined(VINLINE_VATOM)
00114
00121     VEXTERNC double* Vatom_getPosition(Vatom *thee);
00122
00129     VEXTERNC void      Vatom_setRadius(Vatom *thee, double radius);
00130
00137     VEXTERNC double    Vatom_getRadius(Vatom *thee);
00138
00146     VEXTERNC void      Vatom_setPartID(Vatom *thee, int partID);
00147
00155     VEXTERNC double    Vatom_getPartID(Vatom *thee);
00156
00163     VEXTERNC void      Vatom_setAtomID(Vatom *thee, int id);
00164
00171     VEXTERNC double    Vatom_getAtomID(Vatom *thee);
00172
00179     VEXTERNC void      Vatom_setCharge(Vatom *thee, double charge);
00180
00187     VEXTERNC double    Vatom_getCharge(Vatom *thee);
00188
00195     VEXTERNC void      Vatom_setEpsilon(Vatom *thee, double epsilon);
00196
00203     VEXTERNC double    Vatom_getEpsilon(Vatom *thee);
```

```

00204
00212     VEXTERNC unsigned long int Vatom_memChk(Vatom *thee);
00213
00214 #else /* if defined(VINLINE_VATOM) */
00215 #   define Vatom_getPosition(thee) ((thee)->position)
00216 #   define Vatom_setRadius(thee, tRadius) ((thee)->radius = (tRadius))
00217 #   define Vatom_getRadius(thee) ((thee)->radius)
00218 #   define Vatom_setPartID(thee, tpartID) ((thee)->partID = (double)(tpartID))
00219 #   define Vatom_getPartID(thee) ((thee)->partID)
00220 #   define Vatom_setAtomID(thee, tatomID) ((thee)->id = (tatomID))
00221 #   define Vatom_getAtomID(thee) ((thee)->id)
00222 #   define Vatom_setCharge(thee, tCharge) ((thee)->charge = (tCharge))
00223 #   define Vatom_getCharge(thee) ((thee)->charge)
00224 #   define Vatom_setEpsilon(thee, tEpsilon) ((thee)->epsilon = (tEpsilon))
00225 #   define Vatom_getEpsilon(thee) ((thee)->epsilon)
00226 #   define Vatom_memChk(thee) (sizeof(Vatom))
00227 #endif /* if !defined(VINLINE_VATOM) */
00228
00229 /* //////////////////////////////// Class Vatom: Non-Inlineable methods (vatom.c)
00230
00232
00239 VEXTERNC void      Vatom_setResName(Vatom *thee, char resName[VMAX_RECLEN]);
00240
00245 VEXTERNC void      Vatom_setAtomName(
00246     Vatom *thee,
00247     char atomName[VMAX_RECLEN]
00248 );
00249
00256 VEXTERNC void      Vatom_getResName(Vatom *thee, char resName[VMAX_RECLEN]);
00257
00262 VEXTERNC void      Vatom_getAtomName(
00263     Vatom *thee,
00264     char atomName[VMAX_RECLEN]
00265 );
00266
00272 VEXTERNC Vatom* Vatom_ctor();
00273
00280 VEXTERNC int       Vatom_ctor2(Vatom *thee);
00281
00287 VEXTERNC void      Vatom_dtor(Vatom **thee);
00288
00294 VEXTERNC void      Vatom_dtor2(Vatom *thee);
00295
00302 VEXTERNC void      Vatom_setPosition(Vatom *thee, double position[3]);
00303
00311 VEXTERNC void Vatom_copyTo(Vatom *thee, Vatom *dest);
00312
00320 VEXTERNC void Vatom_copyFrom(Vatom *thee, Vatom *src);
00321
00322 #if defined(WITH_TINKER)
00323
00330 VEXTERNC void      Vatom_setInducedDipole(Vatom *thee,
00331                                         double inducedDipole[3]);
00332
00339 VEXTERNC void      Vatom_setNLInducedDipole(Vatom *thee,
00340                                         double nlInducedDipole[3]);
00341

```

```

00348 VEXTERNC void    Vatom_setDipole(Vatom *thee, double dipole[3]);
00349
00356 VEXTERNC void    Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]);
00357
00363 VEXTERNC double*  Vatom_getDipole(Vatom *thee);
00364
00370 VEXTERNC double*  Vatom_getQuadrupole(Vatom *thee);
00371
00377 VEXTERNC double*  Vatom_getInducedDipole(Vatom *thee);
00378
00384 VEXTERNC double*  Vatom_getNLInducedDipole(Vatom *thee);
00385 #endif /* if defined(WITH_TINKER) */
00386
00387 #endif /* ifndef _VATOM_H_ */

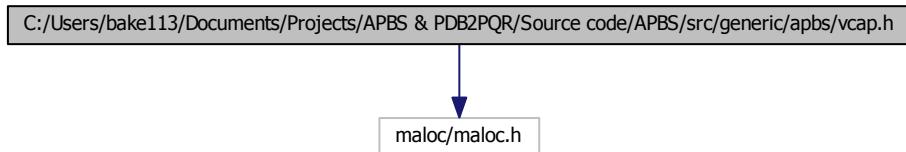
```

10.35 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vcap.h File Reference

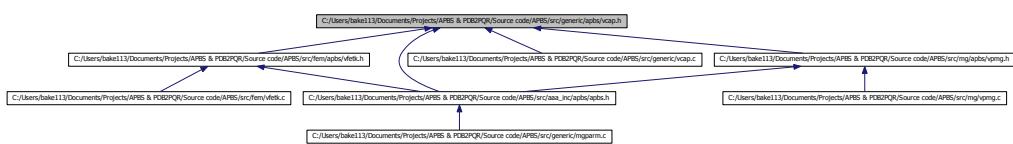
Contains declarations for class Vcap.

```
#include "maloc/maloc.h"
```

Include dependency graph for vcap.h:



This graph shows which files directly or indirectly include this file:



Defines

- `#define EXPMAX 85.00`
Maximum argument for exp(), sinh(), or cosh()
- `#define EXPMIN -85.00`
Minimum argument for exp(), sinh(), or cosh()

Functions

- VEXTERNC double `Vcap_exp` (double x, int *ichop)
Provide a capped exp() function.
- VEXTERNC double `Vcap_sinh` (double x, int *ichop)
Provide a capped sinh() function.
- VEXTERNC double `Vcap_cosh` (double x, int *ichop)
Provide a capped cosh() function.

10.35.1 Detailed Description

Contains declarations for class Vcap.

Version

Id:

`vcap.h` 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
```

```
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [vcap.h](#).

**10.36 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source
code/APBS/src/generic/apbs/vcap.h**

```
00001  
00064 #ifndef _VCAP_H_  
00065 #define _VCAP_H_  
00066  
00070 #define EXPMAX 85.00  
00071  
00075 #define EXPMIN -85.00  
00076  
00077 #include "maloc/maloc.h"  
00078  
00097 VEXTERNC double Vcap_exp(  
00098     double x,  
00099     int *ichop  
00100 );
```

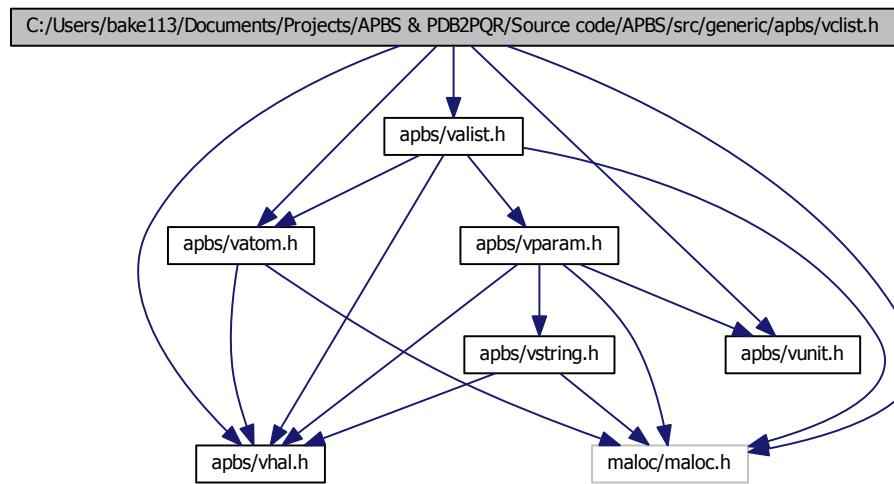
```
00101
00102
00121 VEXTERNC double Vcap_sinh(
00122     double x,
00123     int *ichop
00124 );
00125
00144 VEXTERNC double Vcap_cosh(
00145     double x,
00146     int *ichop
00147 );
00148
00149 #endif /* ifndef _VCAP_H_ */
```

10.37 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vclist.h File Reference

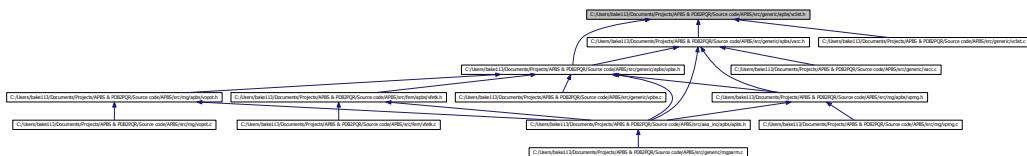
Contains declarations for class Vclist.

```
#include "malloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/valist.h"
#include "apbs/vatom.h"
#include "apbs/vunit.h"
```

Include dependency graph for vclist.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sVclistCell`
Atom cell list cell.
- struct `sVclist`
Atom cell list.

Typedefs

- typedef enum `eVclist_DomainMode` `Vclist_DomainMode`

Declaration of Vclist_DomainMode enumeration type.

- **typedef struct sVclistCell VclistCell**

Declaration of the VclistCell class as the VclistCell structure.

- **typedef struct sVclist Vclist**

Declaration of the Vclist class as the Vclist structure.

Enumerations

- enum **eVclist_DomainMode { CLIST_AUTO_DOMAIN, CLIST_MANUAL_DOMAIN }**

Atom cell list domain setup mode.

Functions

- VEXTERNC unsigned long int **Vclist_memChk (Vclist *thee)**
Get number of bytes in this object and its members.
- VEXTERNC double **Vclist_maxRadius (Vclist *thee)**
Get the max probe radius value (in A) the cell list was constructed with.
- VEXTERNC **Vclist * Vclist_ctor (Valist *alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])**
Construct the cell list object.
- VEXTERNC **Vrc_Codes Vclist_ctor2 (Vclist *thee, Valist *alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])**
FORTRAN stub to construct the cell list object.
- VEXTERNC void **Vclist_dtor (Vclist **thee)**
Destroy object.
- VEXTERNC void **Vclist_dtor2 (Vclist *thee)**
FORTRAN stub to destroy object.
- VEXTERNC **VclistCell * Vclist_getCell (Vclist *thee, double position[VAPBS_DIM])**
Return cell corresponding to specified position or return VNULL.
- VEXTERNC **VclistCell * VclistCell_ctor (int natoms)**
Allocate and construct a cell list cell object.
- VEXTERNC **Vrc_Codes VclistCell_ctor2 (VclistCell *thee, int natoms)**
Construct a cell list object.
- VEXTERNC void **VclistCell_dtor (VclistCell **thee)**
Destroy object.
- VEXTERNC void **VclistCell_dtor2 (VclistCell *thee)**
FORTRAN stub to destroy object.

10.37.1 Detailed Description

Contains declarations for class Vclist.

Version

Id:

[vclist.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vclist.h](#).

10.38 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vclist.h

```

00001
00062 #ifndef _VCLIST_H_
00063 #define _VCLIST_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Headers specific to this file */
00070 #include "apbs/valist.h"
00071 #include "apbs/vatom.h"
00072 #include "apbs/vunit.h"
00073
00079 enum eVclist_DomainMode {
00080     CLIST_AUTO_DOMAIN,
00082     CLIST_MANUAL_DOMAIN
00084 };
00085
00091 typedef enum eVclist_DomainMode Vclist_DomainMode;
00092
00098 struct sVclistCell {
00099     Vatom **atoms;
00100     int natoms;
00101 };
00102
00107 typedef struct sVclistCell VclistCell;
00108
00114 struct sVclist {
00115
00116     Vmem *vmem;
00117     Valist *alist;
00118     Vclist_DomainMode mode;
00119     int npts[VAPBS_DIM];
00120     int n;
00121     double max_radius;
00122     VclistCell *cells;
00123     double lower_corner[VAPBS_DIM];
00124     double upper_corner[VAPBS_DIM];
00125     double spacs[VAPBS_DIM];

```

```
00127 };
00128
00133 typedef struct sVclist Vclist;
00134
00135 #if !defined(VINLINE_VCLIST)
00136
00142     VEXTERNC unsigned long int Vclist_memChk(
00143         Vclist *thee
00144     );
00145
00153     VEXTERNC double Vclist_maxRadius(
00154         Vclist *thee
00155     );
00156
00157 #else /* if defined(VINLINE_VCLIST) */
00158
00159 # define Vclist_memChk(thee) (Vmem_bytes((thee)->vmem))
00160 # define Vclist_maxRadius(thee) ((thee)->max_radius)
00161
00162 #endif /* if !defined(VINLINE_VCLIST) */
00163
00164 /* //////////////////////////////// Class Vclist: Non-Inlineable methods (vclist.c)
00165
00167
00172 VEXTERNC Vclist* Vclist_ctor(
00173     Valist *alist,
00174     double max_radius,
00175     int npts[VAPBS_DIM],
00177     Vclist_DomainMode mode,
00178     double lower_corner[VAPBS_DIM],
00181     double upper_corner[VAPBS_DIM]
00184     );
00185
00186
00190 VEXTERNC Vrc_Codes Vclist_ctor2(
00191     Vclist *thee,
00192     Valist *alist,
00193     double max_radius,
00194     int npts[VAPBS_DIM],
00196     Vclist_DomainMode mode,
00197     double lower_corner[VAPBS_DIM],
00200     double upper_corner[VAPBS_DIM]
00203     );
00204
00209 VEXTERNC void Vclist_dtor(
00210     Vclist **thee
00211     );
00212
00217 VEXTERNC void Vclist_dtor2(
00218     Vclist *thee
00219     );
00220
00228 VEXTERNC VclistCell* Vclist_getCell(
00229     Vclist *thee,
00230     double position[VAPBS_DIM]
00231     );
00232
00239 VEXTERNC VclistCell* VclistCell_ctor(
```

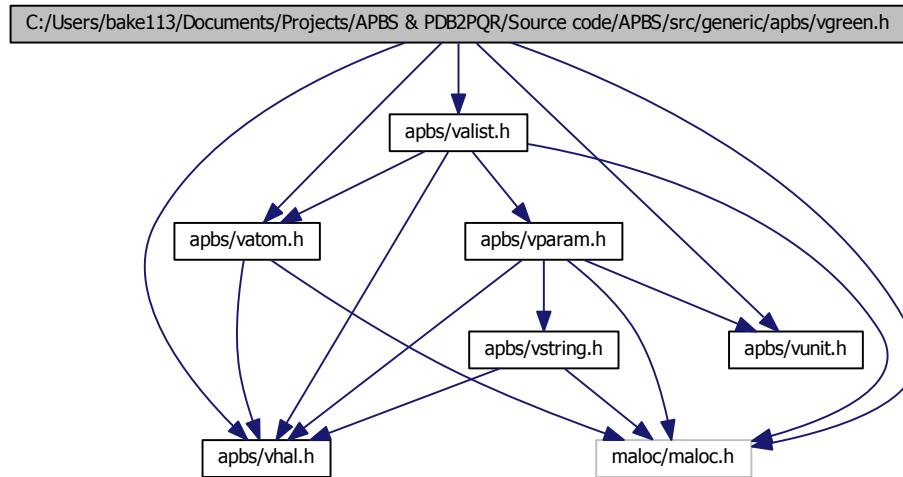
```
00240     int natoms
00241 );
00242
00249 VEXTERNC Vrc_Codes VclistCell_ctor2(
00250     VclistCell *thee,
00251     int natoms
00252 );
00253
00258 VEXTERNC void VclistCell_dtor(
00259     VclistCell **thee
00260 );
00261
00266 VEXTERNC void VclistCell_dtor2(
00267     VclistCell *thee
00268 );
00269
00270 #endif /* ifndef _VCLIST_H_ */
```

10.39 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vgreen.h File Reference

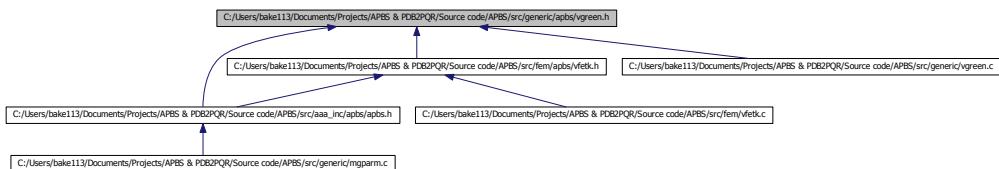
Contains declarations for class Vgreen.

```
#include "malloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vunit.h"
#include "apbs/vatom.h"
#include "apbs/valist.h"
```

Include dependency graph for vgreen.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVgreen](#)
Contains public data members for Vgreen class/module.

TypeDefs

- typedef struct [sVgreen](#) [Vgreen](#)

Declaration of the Vgreen class as the Vgreen structure.

Functions

- VEXTERNC `Valist * Vgreen_getValist (Vgreen *thee)`
Get the atom list associated with this Green's function object.
- VEXTERNC unsigned long int `Vgreen_memChk (Vgreen *thee)`
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC `Vgreen * Vgreen_ctor (Valist *alist)`
Construct the Green's function oracle.
- VEXTERNC int `Vgreen_ctor2 (Vgreen *thee, Valist *alist)`
FORTRAN stub to construct the Green's function oracle.
- VEXTERNC void `Vgreen_dtor (Vgreen **thee)`
Destruct the Green's function oracle.
- VEXTERNC void `Vgreen_dtor2 (Vgreen *thee)`
FORTRAN stub to destruct the Green's function oracle.
- VEXTERNC int `Vgreen_helmholtz (Vgreen *thee, int npos, double *x, double *y, double *z, double *val, double kappa)`
Get the Green's function for Helmholtz's equation integrated over the atomic point charges.
- VEXTERNC int `Vgreen_helmholtzD (Vgreen *thee, int npos, double *x, double *y, double *z, double *gradx, double *grady, double *gradz, double kappa)`
Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.
- VEXTERNC int `Vgreen_coulomb_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)`
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VEXTERNC int `Vgreen_coulomb (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)`
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)
- VEXTERNC int `Vgreen_coulombD (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)`
Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VEXTERNC int `Vgreen_coulombD (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)`
Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. Johnston/R. Krasny FMM library (if available)

10.39.1 Detailed Description

Contains declarations for class Vgreen.

Version

Id:

[vgreen.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Definition in file [vgreen.h](#).

10.40 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vgreen.h

```
00001
00065 #ifndef _VGREEN_H_
00066 #define _VGREEN_H_
00067
00068 /* Generic headers */
00069 #include "maloc/maloc.h"
00070 #include "apbs/vhal.h"
00071
00072 /* Specific headers */
00073 #include "apbs/vunit.h"
00074 #include "apbs/vatom.h"
00075 #include "apbs/valist.h"
00076
00077
00083 struct sVgreen {
00084
00085     Valist *alist;
00086     Vmem *vmem;
00087     double *xp;
00088     double *yp;
00089     double *zp;
00090     double *qp;
00091     int np;
00092 };
00093
00102 typedef struct sVgreen Vgreen;
00103
00104 /* //////////////////////////////// Class Vgreen: Inlineable methods (vgreen.c)
00105 // Class Vgreen: Inlineable methods (vgreen.c)
00106
00107
```

```

00108 #if !defined(VINLINE_VGREEN)
00109
00117     VEXTERNC Valist* Vgreen_getValist(Vgreen *thee);
00118
00126     VEXTERNC unsigned long int Vgreen_memChk(Vgreen *thee);
00127
00128 #else /* if defined(VINLINE_VGREEN) */
00129 #    define Vgreen_getValist(thee) ((thee)->alist)
00130 #    define Vgreen_memChk(thee) (Vmem_bytes((thee)->vmem))
00131 #endif /* if !defined(VINLINE_VGREEN) */
00132
00133 /* //////////////////////////////// */
00134 // Class Vgreen: Non-Inlineable methods (vgreen.c)
00136
00143 VEXTERNC Vgreen* Vgreen_ctor(Valist *alist);
00144
00152 VEXTERNC int Vgreen_ctor2(Vgreen *thee, Valist *alist);
00153
00159 VEXTERNC void Vgreen_dtor(Vgreen **thee);
00160
00166 VEXTERNC void Vgreen_dtor2(Vgreen *thee);
00167
00192 VEXTERNC int Vgreen_helmholtz(Vgreen *thee, int npos, double *x, double *y,
00193     double *z, double *val, double kappa);
00194
00222 VEXTERNC int Vgreen_helmholtzD(Vgreen *thee, int npos, double *x, double *y,
00223     double *z, double *gradx, double *grady, double *gradz, double kappa);
00224
00245 VEXTERNC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
00246     double *y, double *z, double *val);
00247
00268 VEXTERNC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
00269     double *z, double *val);
00270
00294 VEXTERNC int Vgreen_coulombD_direct(Vgreen *thee, int npos, double *x,
00295     double *y, double *z, double *pot, double *gradx, double *grady, double
00296     *gradz);
00297
00322 VEXTERNC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
00323     double *z, double *pot, double *gradx, double *grady, double *gradz);
00324
00325 #endif /* ifndef _VGREEN_H_ */

```

10.41 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vhal.h File Reference

Contains generic macro definitions for APBS.

This graph shows which files directly or indirectly include this file:



Defines

- #define APBS_TIMER_WALL_CLOCK 26
 - APBS total execution timer ID.*
- #define APBS_TIMER_SETUP 27
 - APBS setup timer ID.*
- #define APBS_TIMER_SOLVER 28
 - APBS solver timer ID.*
- #define APBS_TIMER_ENERGY 29
 - APBS energy timer ID.*
- #define APBS_TIMER_FORCE 30
 - APBS force timer ID.*
- #define APBS_TIMER_TEMP1 31
 - APBS temp timer #1 ID.*
- #define APBS_TIMER_TEMP2 32
 - APBS temp timer #2 ID.*
- #define MAXMOL 5
 - The maximum number of molecules that can be involved in a single PBE calculation.*
- #define MAXION 10
 - The maximum number of ion species that can be involved in a single PBE calculation.*
- #define MAXFOCUS 5
 - The maximum number of times an MG calculation can be focused.*
- #define VMGNLEV 4
 - Minimum number of levels in a multigrid calculations.*
- #define VREDFRAC 0.25
 - Maximum reduction of grid spacing during a focusing calculation.*
- #define VAPBS_NVS 4
 - Number of vertices per simplex (hard-coded to 3D)*
- #define VAPBS_DIM 3
 - Our dimension.*
- #define VAPBS_RIGHT 0
 - Face definition for a volume.*
- #define MAX_SPHERE PTS 50000

- **#define VAPBS_FRONT 1**
Face definition for a volume.
- **#define VAPBS_UP 2**
Face definition for a volume.
- **#define VAPBS_LEFT 3**
Face definition for a volume.
- **#define VAPBS_BACK 4**
Face definition for a volume.
- **#define VAPBS_DOWN 5**
Face definition for a volume.
- **#define VPMGSMALL 1e-12**
A small number used in Vpmg to decide if points are on/off grid-lines or non-zero (etc.)
- **#define SINH_MIN -85.0**
Used to set the min values acceptable for sinh chopping.
- **#define SINH_MAX 85.0**
Used to set the max values acceptable for sinh chopping.
- **#define VF77_MANGLE(name, NAME) name**
Name-mangling macro for using FORTRAN functions in C code.
- **#define VFLOOR(value) floor(value)**
Wrapped floor to fix floating point issues in the Intel compiler.
- **#define VEMBED(rctag)**
Allows embedding of RCS ID tags in object files.

Typedefs

- **typedef enum eVrc_Codes Vrc_Codes**
Declaration of the Vrc_Codes type as the Vrc_Codes enum.
- **typedef enum eVsol_Meth Vsol_Meth**
Declaration of the Vsol_Meth type as the Vsol_Meth enum.
- **typedef enum eVsurf_Meth Vsurf_Meth**
Declaration of the Vsurf_Meth type as the Vsurf_Meth enum.
- **typedef enum eVhal_PBEType Vhal_PBEType**
Declaration of the Vhal_PBEType type as the Vhal_PBEType enum.
- **typedef enum eVhal_IPKEYType Vhal_IPKEYType**
Declaration of the Vhal_IPKEYType type as the Vhal_IPKEYType enum.
- **typedef enum eVhal_NONLINType Vhal_NONLINType**
Declaration of the Vhal_NONLINType type as the Vhal_NONLINType enum.
- **typedef enum eVoutput_Format Voutput_Format**
Declaration of the Voutput_Format type as the VOutput_Format enum.
- **typedef enum eVbcfl Vbcfl**

Declare Vbcfl type.

- **typedef enum eVchrg_Meth Vchrg_Meth**
Declaration of the Vchrg_Meth type as the Vchrg_Meth enum.
- **typedef enum eVchrg_Src Vchrg_Src**
Declaration of the Vchrg_Src type as the Vchrg_Meth enum.
- **typedef enum eVdata_Type Vdata_Type**
Declaration of the Vdata_Type type as the Vdata_Type enum.
- **typedef enum eVdata_Format Vdata_Format**
Declaration of the Vdata_Format type as the Vdata_Format enum.

Enumerations

- **enum eVrc_Codes { VRC_WARNING = -1, VRC_FAILURE = 0, VRC_SUCCESS = 1 }**
Return code enumerations.
- **enum eVsol_Meth {
VSOL_CGMG, VSOL_Newton, VSOL_MG, VSOL(CG,
VSOL_SOR, VSOL_RBGS, VSOL_WJ, VSOL_Richardson,
VSOL_CGMGAqua, VSOL_NewtonAqua }**
Solution Method enumerations.
- **enum eVsurf_Meth {
VSM_MOL = 0, VSM_MOLSMOOTH = 1, VSM_SPLINE = 2, VSM_SPLINE3 =
3,
VSM_SPLINE4 = 4 }**
Types of molecular surface definitions.
- **enum eVhal_PBEType {
PBE_LPBE, PBE_NPBE, PBE_LRPBE, PBE_NRPBE,
PBE_SMPBE }**
Version of PBE to solve.
- **enum eVhal_IPKEYType { IPKEY_SMPBE = -2, IPKEY_LPBE, IPKEY_NPBE }**
Type of ipkey to use for MG methods.
- **enum eVhal_NONLINType {
NONLIN_LPBE = 0, NONLIN_NPBE, NONLIN_SMPBE, NONLIN_LPBEAQUA,
NONLIN_NPBEAQUA }**
Type of nonlinear to use for MG methods.
- **enum eVoutput_Format { OUTPUT_NULL, OUTPUT_FLAT }**
Output file format.

- enum eVbcfl {

BCFL_ZERO = 0, BCFL_SDH = 1, BCFL_MDH = 2, BCFL_UNUSED = 3,

BCFL_FOCUS = 4, BCFL_MEM = 5, BCFL_MAP = 6 }

Types of boundary conditions.
- enum eVchrg_Meth { VCM_TRI1 = 0, VCM_BSPL2 = 1, VCM_BSPL4 = 2 }

Types of charge discretization methods.
- enum eVchrg_Src { VCM_CHARGE = 0, VCM_PERMANENT = 1, VCM_INDUCED = 2, VCM_NLINDUCED = 3 }

Charge source.
- enum eVdata_Type {

VDT_CHARGE, VDT_POT, VDT_ATOMPOT, VDT_SMOL,

VDT_SSPL, VDT_VDW, VDT_IVDW, VDT_LAP,

VDT_EDENS, VDT_NDENS, VDT_QDENS, VDT_DIELX,

VDT_DIELY, VDT_DIELZ, VDT_KAPPA }

Types of (scalar) data that can be written out of APBS.
- enum eVdata_Format {

VDF_DX = 0, VDF_UHBD = 1, VDF_AVIS = 2, VDF_MCSF = 3,

VDF_GZ = 4, VDF_FLAT = 5 }

Format of data for APBS I/O.

10.41.1 Detailed Description

Contains generic macro definitions for APBS.

Version

Id:

vhal.h 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
```

```
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [vhal.h](#).

10.42 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vhal.h

```
00001  
00064 #ifndef _VAPBSHAL_H_  
00065 #define _VAPBSHAL_H_  
00066  
00073 enum eVrc_Codes {  
00074     VRC_WARNING=-1,  
00075
```

```
00076 VRC_FAILURE=0,
00077 VRC_SUCCESS=1
00079 };
00080 typedef enum eVrc_Codes Vrc_Codes;
00081
00082 enum eVsol_Meth {
00083
00084 VSOL_CGMG, /* 0: conjugate gradient multigrid */
00085 VSOL_Newton, /* 1: newton */
00086 VSOL_MG, /* 2: multigrid */
00087 VSOL(CG, /* 3: conjugate gradient */
00088 VSOL_SOR, /* 4: sucessive overrelaxation */
00089 VSOL_RBGS, /* 5: red-black gauss-seidel */
00090 VSOL_WJ, /* 6: weighted jacobi */
00091 VSOL_Richardson, /* 7: richardson */
00092 VSOL_CGMGAqua, /* 8: conjugate gradient multigrid aqua */
00093 VSOL_NewtonAqua /* 9: newton aqua */
00094
00095 };
00096 typedef enum eVsol_Meth Vsol_Meth;
00097
00098 enum eVsurf_Meth {
00099 VSM_MOL=0,
00100 VSM_MOLSMOOTH=1,
00101 VSM_SPLINE=2,
00102 VSM_SPLINE3=3,
00103 VSM_SPLINE4=4
00104 };
00105
00106 typedef enum eVsurf_Meth Vsurf_Meth;
00107
00108 enum eVhal_PBEType {
00109 PBE_LPBE,
00110 PBE_NPBE,
00111 PBE_LRPBE,
00112 PBE_NRPBE,
00113 PBE_SMPBE
00114 };
00115
00116 typedef enum eVhal_PBEType Vhal_PBEType;
00117
00118 enum eVhal_IPKEYType {
00119 IPKEY_SMPBE = -2,
00120 IPKEY_LPBE,
00121 IPKEY_NPBE
00122 };
00123
00124 typedef enum eVhal_IPKEYType Vhal_IPKEYType;
00125
00126 enum eVhal_NONLINType {
00127 NONLIN_LPBE = 0,
00128 NONLIN_NPBE,
00129 NONLIN_SMPBE,
00130 NONLIN_LPBEAQUA,
00131 NONLIN_NPBEAQUA
00132 };
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
```

```
00192 typedef enum eVhal_NONLINType Vhal_NONLINType;
00193
00198 enum eVoutput_Format {
00199     OUTPUT_NULL,
00200     OUTPUT_FLAT,
00201 };
00202
00207 typedef enum eVoutput_Format Voutput_Format;
00208
00214 enum eVbcfl {
00215     BCFL_ZERO=0,
00216     BCFL_SDH=1,
00218     BCFL_MDH=2,
00220     BCFL_UNUSED=3,
00221     BCFL_FOCUS=4,
00222     BCFL_MEM=5,
00223     BCFL_MAP=6
00224 };
00225
00230 typedef enum eVbcfl Vbcfl;
00231
00237 enum eVchrg_Meth {
00238     VCM_TRI=0,
00241     VCM_BSPL2=1,
00244     VCM_BSPL4=2
00245 };
00246
00251 typedef enum eVchrg_Meth Vchrg_Meth;
00252
00258 enum eVchrg_Src {
00259     VCM_CHARGE=0,
00260     VCM_PERMANENT=1,
00261     VCM_INDUCED=2,
00262     VCM_NLINDUCED=3
00263 };
00264
00269 typedef enum eVchrg_Src Vchrg_Src;
00270
00276 enum eVdata_Type {
00277     VDT_CHARGE,
00278     VDT_POT,
00279     VDT_ATOMPOT,
00280     VDT_SMOL,
00282     VDT_SSPL,
00284     VDT_VDW,
00286     VDT_IVDW,
00288     VDT_LAP,
00289     VDT_EDENS,
00291     VDT_NDENS,
00293     VDT_QDENS,
00295     VDT_DIELX,
00297     VDT_DIELY,
00299     VDT_DIELZ,
00301     VDT_KAPPA
00303 };
00304
00309 typedef enum eVdata_Type Vdata_Type;
```

```
00310
00316 enum eVdata_Format {
00317     VDF_DX=0,
00318     VDF_UHBD=1,
00319     VDF_AVIS=2,
00320     VDF_MCSF=3,
00321     VDF_GZ=4,
00322     VDF_FLAT=5
00323 };
00324
00329 typedef enum eVdata_Format Vdata_Format;
00330
00335 #define APBS_TIMER_WALL_CLOCK 26
00336
00341 #define APBS_TIMER_SETUP 27
00342
00347 #define APBS_TIMER_SOLVER 28
00348
00353 #define APBS_TIMER_ENERGY 29
00354
00359 #define APBS_TIMER_FORCE 30
00360
00365 #define APBS_TIMER_TEMP1 31
00366
00371 #define APBS_TIMER_TEMP2 32
00372
00377 #define MAXMOL 5
00378
00383 #define MAXION 10
00384
00388 #define MAXFOCUS 5
00389
00393 #define VMGNLEV 4
00394
00398 #define VREDFRAC 0.25
00399
00403 #define VAPBS_NVS 4
00404
00408 #define VAPBS_DIM 3
00409
00414 #define VAPBS_RIGHT 0
00415
00420 #define MAX_SPHERE PTS 50000
00421
00426 #define VAPBS_FRONT 1
00427
00432 #define VAPBS_UP 2
00433
00438 #define VAPBS_LEFT 3
00439
00444 #define VAPBS_BACK 4
00445
00450 #define VAPBS_DOWN 5
00451
00456 #define VPMGSMALL 1e-12
00457
00462 #define SINH_MIN -85.0
```

```
00463
00468 #define SINH_MAX 85.0
00469
00470
00471 #if defined(VDEBUG)
00472 #  if !defined(APBS_NOINLINE)
00473 #    define APBS_NOINLINE 1
00474 #  endif
00475 #endif
00476
00477 #if !defined(APBS_NOINLINE)
00478
00482 #  define VINLINE_VACC
00483
00487 #  define VINLINE_VATOM
00488
00492 #  define VINLINE_VCSM
00493
00497 #  define VINLINE_VPBE
00498
00502 #  define VINLINE_VPEE
00503
00507 #  define VINLINE_VGREEN
00508
00512 #  define VINLINE_VFETK
00513
00517 #  define VINLINE_VPMG
00518
00523 #  define MAX_HASH_DIM 75
00524
00525 #endif
00526
00527 /* Fortran name mangling */
00528 #if defined(VF77_UPPERCASE)
00529 #  if defined(VF77_NOUNDERSCORE)
00530 #    define VF77_MANGLE(name,NAME) NAME
00531 #  elif defined(VF77_ONEUNDERSCORE)
00532 #    define VF77_MANGLE(name,NAME) NAME ## _
00533 #  else
00534 #    define VF77_MANGLE(name,NAME) name
00535 #  endif
00536 #else
00537 #  if defined(VF77_NOUNDERSCORE)
00538 #    define VF77_MANGLE(name,NAME) name
00539 #  elif defined(VF77_ONEUNDERSCORE)
00540 #    define VF77_MANGLE(name,NAME) name ## _
00541 #  else
00542
00545 #    define VF77_MANGLE(name,NAME) name
00546 #  endif
00547 #endif
00548
00549 /* Floating Point Error */
00550 #if defined(MACHINE_EPS)
00551 #  define VFLOOR(value) \
00552          ((floor(value) != floor(value + MACHINE_EPS)) ? \
00553                  floor(value + MACHINE_EPS) : floor(value))
```

```

00554 #else
00555
00560 #define VFLOOR(value) floor(value)
00561 #endif
00562
00563 /* String embedding for ident */
00564 #if defined(HAVE_EMBED)
00565
00569 #define VEMBED(rctag) \
00570     VPRIIVATE const char* rctag; \
00571     static void* use_rcsid=(0 ? &use_rcsid : (void**)&rcsid);
00572 #else
00573
00577 #define VEMBED(rctag)
00578 #endif /* if defined(HAVE_EMBED) */
00579
00580#endif /* #ifndef _VAPBSHAL_H_ */

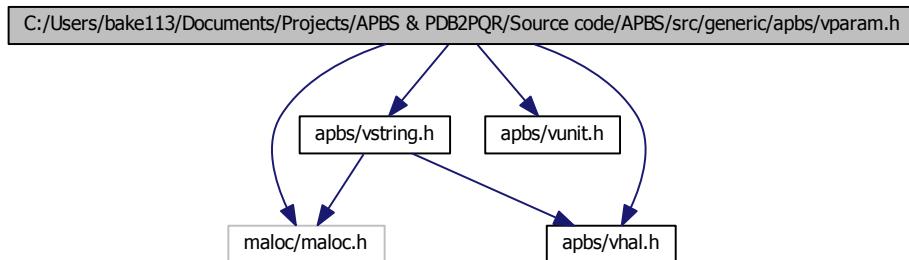
```

10.43 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vparam.h File Reference

Contains declarations for class [Vparam](#).

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vunit.h"
#include "apbs/vstring.h"
```

Include dependency graph for vparam.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVparam_AtomData](#)
AtomData sub-class; stores atom data.
- struct [Vparam_ResData](#)
ResData sub-class; stores residue data.
- struct [Vparam](#)
Reads and assigns charge/radii parameters.

Typedefs

- typedef struct [sVparam_AtomData](#) [Vparam_AtomData](#)
Declaration of the Vparam_AtomData class as the sVparam_AtomData structure.
- typedef struct [Vparam_ResData](#) [Vparam_ResData](#)
Declaration of the Vparam_ResData class as the Vparam_ResData structure.
- typedef struct [Vparam](#) [Vparam](#)
Declaration of the Vparam class as the Vparam structure.

Functions

- VEXTERNC unsigned long int [Vparam_memChk](#) ([Vparam](#) *thee)
Get number of bytes in this object and its members.
- VEXTERNC [Vparam_AtomData](#) * [Vparam_AtomData_ctor](#) ()
Construct the object.
- VEXTERNC int [Vparam_AtomData_ctor2](#) ([Vparam_AtomData](#) *thee)
FORTRAN stub to construct the object.
- VEXTERNC void [Vparam_AtomData_dtor](#) ([Vparam_AtomData](#) **thee)
Destroy object.
- VEXTERNC void [Vparam_AtomData_dtor2](#) ([Vparam_AtomData](#) *thee)
FORTRAN stub to destroy object.

- VEXTERNC void `Vparam_AtomData_copyTo` (`Vparam_AtomData *thee, Vparam_AtomData *dest)`
Copy current atom object to destination.
- VEXTERNC void `Vparam_ResData_copyTo` (`Vparam_ResData *thee, Vparam_ResData *dest)`
Copy current residue object to destination.
- VEXTERNC void `Vparam_AtomData_copyFrom` (`Vparam_AtomData *thee, Vparam_AtomData *src)`
Copy current atom object from another.
- VEXTERNC `Vparam_ResData * Vparam_ResData_ctor` (`Vmem *mem)`
Construct the object.
- VEXTERNC int `Vparam_ResData_ctor2` (`Vparam_ResData *thee, Vmem *mem)`
FORTRAN stub to construct the object.
- VEXTERNC void `Vparam_ResData_dtor` (`Vparam_ResData **thee)`
Destroy object.
- VEXTERNC void `Vparam_ResData_dtor2` (`Vparam_ResData *thee)`
FORTRAN stub to destroy object.
- VEXTERNC `Vparam * Vparam_ctor ()`
Construct the object.
- VEXTERNC int `Vparam_ctor2` (`Vparam *thee)`
FORTRAN stub to construct the object.
- VEXTERNC void `Vparam_dtor` (`Vparam **thee)`
Destroy object.
- VEXTERNC void `Vparam_dtor2` (`Vparam *thee)`
FORTRAN stub to destroy object.
- VEXTERNC `Vparam_ResData * Vparam_getResData` (`Vparam *thee, char resName[VMAX_ARGLEN])`
Get residue data.
- VEXTERNC `Vparam_AtomData * Vparam_getAtomData` (`Vparam *thee, char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN])`
Get atom data.
- VEXTERNC int `Vparam_readFlatFile` (`Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)`
Read a flat-file format parameter database.
- VEXTERNC int `Vparam_readXMLFile` (`Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)`
Read an XML format parameter database.

10.43.1 Detailed Description

Contains declarations for class [Vparam](#).

Version

Id:

[vparam.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vparam.h](#).

10.44 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vparam.h

```

00001
00062 #ifndef _VPARAM_H_
00063 #define _VPARAM_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068 #include "apbs/vunit.h"
00069 #include "apbs/vstring.h"
00070
00071 struct sVparam_AtomData {
00072     char atomName[VMAX_ARGLEN];
00073     char resName[VMAX_ARGLEN];
00074     double charge;
00075     double radius;
00076     double epsilon;
00077 };
00078
00101 typedef struct sVparam_AtomData Vparam_AtomData;
00102
00109 struct Vparam_ResData {
00110     Vmem *vmem;
00111     char name[VMAX_ARGLEN];
00112     int nAtomData;
00113     Vparam_AtomData *atomData;
00114 };
00115
00116
00122 typedef struct Vparam_ResData Vparam_ResData;
00123
00130 struct Vparam {
00131
00132     Vmem *vmem;
00133     int nResData;
00135     Vparam_ResData *resData;
00136 };
00137
00142 typedef struct Vparam Vparam;
00143
00144 /* //////////////////////////////// */

```

```
00145 // Class Vparam: Inlineable methods (vparam.c)
00147
00148 #if !defined(VINLINE_VPARAM)
00149
00156     VEXTERNC unsigned long int Vparam_memChk(Vparam *thee);
00157
00158 #else /* if defined(VINLINE_VPARAM) */
00159
00160 #    define Vparam_memChk(thee) (Vmem_bytes((thee)->vmem))
00161
00162 #endif /* if !defined(VINLINE_VPARAM) */
00163
00164 /* //////////////////////////////// */
00165 // Class Vparam: Non-Inlineable methods (vparam.c)
00167
00172 VEXTERNC Vparam_AtomData* Vparam_AtomData_ctor();
00173
00179 VEXTERNC int Vparam_AtomData_ctor2(Vparam_AtomData *thee);
00180
00185 VEXTERNC void Vparam_AtomData_dtor(Vparam_AtomData **thee);
00186
00191 VEXTERNC void Vparam_AtomData_dtor2(Vparam_AtomData *thee);
00192
00200 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
00201     Vparam_AtomData *dest);
00202
00210 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
00211     Vparam_ResData *dest);
00212
00220 VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
00221     Vparam_AtomData *src);
00222
00228 VEXTERNC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem);
00229
00236 VEXTERNC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem);
00237
00242 VEXTERNC void Vparam_ResData_dtor(Vparam_ResData **thee);
00243
00248 VEXTERNC void Vparam_ResData_dtor2(Vparam_ResData *thee);
00249
00254 VEXTERNC Vparam* Vparam_ctor();
00255
00261 VEXTERNC int Vparam_ctor2(Vparam *thee);
00262
00267 VEXTERNC void Vparam_dtor(Vparam **thee);
00268
00273 VEXTERNC void Vparam_dtor2(Vparam *thee);
00274
00285 VEXTERNC Vparam_ResData* Vparam_getResData(Vparam *thee,
00286     char resName[VMAX_ARGLEN]);
00287
00299 VEXTERNC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
00300     char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]);
00301
00330 VEXTERNC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
00331     const char *iofmt, const char *thost, const char *fname);
00332
```

```

00343 VEXTERNC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
00344     const char *iofmt, const char *thost, const char *fname);
00345
00346 #endif /* ifndef _VPARAM_H_ */

```

10.45 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vpbe.h File Reference

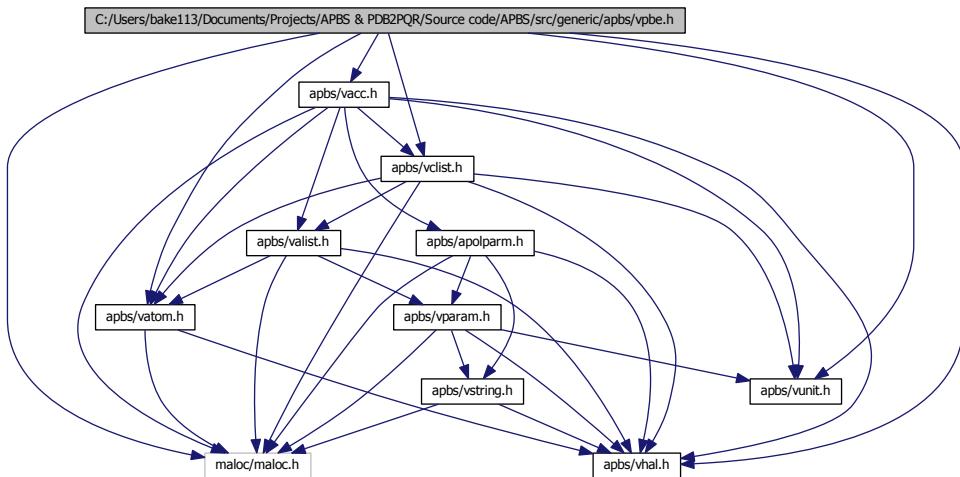
Contains declarations for class Vpbe.

```

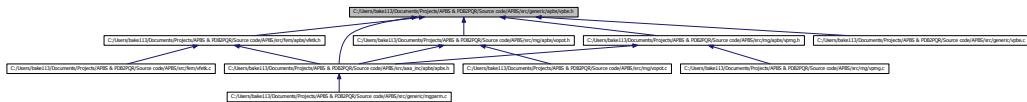
#include "maloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vunit.h"
#include "apbs/vatom.h"
#include "apbs/vacc.h"
#include "apbs/vclist.h"

```

Include dependency graph for vpbe.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVpbe](#)
Contains public data members for Vpbe class/module.

Typedefs

- typedef struct [sVpbe](#) [Vpbe](#)
Declaration of the Vpbe class as the Vpbe structure.

Functions

- VEXTERNC [Valist](#) * [Vpbe_getValist](#) ([Vpbe](#) *thee)
Get atom list.
- VEXTERNC [Vacc](#) * [Vpbe_getVacc](#) ([Vpbe](#) *thee)
Get accessibility oracle.
- VEXTERNC double [Vpbe_getBulkIonicStrength](#) ([Vpbe](#) *thee)
Get bulk ionic strength.
- VEXTERNC double [Vpbe_getMaxIonRadius](#) ([Vpbe](#) *thee)
Get maximum radius of ion species.
- VEXTERNC double [Vpbe_getTemperature](#) ([Vpbe](#) *thee)
Get temperature.
- VEXTERNC double [Vpbe_getSoluteDiel](#) ([Vpbe](#) *thee)
Get solute dielectric constant.
- VEXTERNC double [Vpbe_getGamma](#) ([Vpbe](#) *thee)
Get apolar coefficient.
- VEXTERNC double [Vpbe_getSoluteRadius](#) ([Vpbe](#) *thee)
Get sphere radius which bounds biomolecule.
- VEXTERNC double [Vpbe_getSoluteXlen](#) ([Vpbe](#) *thee)
Get length of solute in x dimension.
- VEXTERNC double [Vpbe_getSoluteYlen](#) ([Vpbe](#) *thee)

- VEXTERNC double `Vpbe_getSoluteZlen` (`Vpbe *thee`)

Get length of solute in y dimension.
- VEXTERNC double * `Vpbe_getSoluteCenter` (`Vpbe *thee`)

Get length of solute in z dimension.
- VEXTERNC double `Vpbe_getSoluteCharge` (`Vpbe *thee`)

Get coordinates of solute center.
- VEXTERNC double `Vpbe_getSolventDiel` (`Vpbe *thee`)

Get total solute charge.
- VEXTERNC double `Vpbe_getSolventRadius` (`Vpbe *thee`)

Get solvent dielectric constant.
- VEXTERNC double `Vpbe_getXkappa` (`Vpbe *thee`)

Get solvent molecule radius.
- VEXTERNC double `Vpbe_getXkappa2` (`Vpbe *thee`)

Get Debye-Hückel parameter.
- VEXTERNC double `Vpbe_getDeblen` (`Vpbe *thee`)

Get Debye-Hückel screening length.
- VEXTERNC double `Vpbe_getZkappa2` (`Vpbe *thee`)

Get modified squared Debye-Hückel parameter.
- VEXTERNC double `Vpbe_getZmagic` (`Vpbe *thee`)

Get charge scaling factor.
- VEXTERNC double `Vpbe_getzmem` (`Vpbe *thee`)

Get z position of the membrane bottom.
- VEXTERNC double `Vpbe_getLmem` (`Vpbe *thee`)

*Get length of the membrane (A)
author Michael Grabe.*
- VEXTERNC double `Vpbe_getmembraneDiel` (`Vpbe *thee`)

Get membrane dielectric constant.
- VEXTERNC double `Vpbe_getmemv` (`Vpbe *thee`)

Get membrane potential (kT)
- VEXTERNC `Vpbe *` `Vpbe_ctor` (`Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V`)

Construct Vpbe object.
- VEXTERNC int `Vpbe_ctor2` (`Vpbe *thee, Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V`)

FORTRAN stub to construct Vpbe objct.
- VEXTERNC int `Vpbe_getIons` (`Vpbe *thee, int *nion, double ionConc[MAXION], double ionRadii[MAXION], double ionQ[MAXION]`)

Get information about the counterion species present.

- VEXTERNC void [Vpbe_dtor](#) ([Vpbe](#) **thee)
Object destructor.
- VEXTERNC void [Vpbe_dtor2](#) ([Vpbe](#) *thee)
FORTRAN stub object destructor.
- VEXTERNC double [Vpbe_getCoulombEnergy1](#) ([Vpbe](#) *thee)
Calculate coulombic energy of set of charges.
- VEXTERNC unsigned long int [Vpbe_memChk](#) ([Vpbe](#) *thee)
Return the memory used by this structure (and its contents) in bytes.

10.45.1 Detailed Description

Contains declarations for class [Vpbe](#).

Version

Id:

[vpbe.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.
```

```

*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vpbe.h](#).

10.46 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vpbe.h

```

00001
00066 #ifndef _VPBE_H_
00067 #define _VPBE_H_
00068
00069 /* Generic headers */
00070 #include "malloc/malloc.h"
00071 #include "apbs/vhal.h"
00072
00073 /* Specific headers */
00074 #include "apbs/vunit.h"
00075 #include "apbs/vatom.h"
00076 #include "apbs/vacc.h"
00077 #include "apbs/vclist.h"
00078
00084 struct sVpbe {
00085
00086   Vmem *vmem;
00088   Valist *alist;
00089   Vclist *clist;
00090   Vacc *acc;
00092   double T;
00093   double soluteDiel;
00094   double solventDiel;
00095   double solventRadius;
00099   double bulkIonicStrength;

```

```

00100     double maxIonRadius;
00103     int numIon;
00104     double ionConc[MAXION];
00105     double ionRadii[MAXION];
00106     double ionQ[MAXION];
00108     double xkappa;
00109     double deblen;
00110     double zkappa2;
00111     double zmagic;
00113     double soluteCenter[3];
00114     double soluteRadius;
00115     double soluteXlen;
00116     double soluteYlen;
00117     double soluteZlen;
00118     double soluteCharge;
00120     double smvolume;
00121     double smsize;
00122     int ipkey;
00125     int paramFlag;
00127     /*-----*/
00128     /* Added by Michael Grabe */
00129     /*-----*/
00130
00131     double z_mem;
00132     double L;
00133     double membraneDiel;
00134     double V;
00135     int param2Flag;
00136     /*-----*/
00137
00138 };
00139
00144 typedef struct sVpbe Vpbe;
00145
00146 /* //////////////////////////////// */
00147     // Class Vpbe: Inlineable methods (vpbe.c)
00149
00150 #if !defined(VINLINE_VPBE)
00151
00158 VEXTERNC Valist* Vpbe_getValist(Vpbe *thee);
00159
00166 VEXTERNC Vacc* Vpbe_getVacc(Vpbe *thee);
00167
00174 VEXTERNC double Vpbe_getBulkIonicStrength(Vpbe *thee);
00175
00182 VEXTERNC double Vpbe_getMaxIonRadius(Vpbe *thee);
00183
00190 VEXTERNC double Vpbe_getTemperature(Vpbe *thee);
00191
00198 VEXTERNC double Vpbe_getSoluteDiel(Vpbe *thee);
00199
00206 VEXTERNC double Vpbe_getGamma(Vpbe *thee);
00207
00214 VEXTERNC double Vpbe_getSoluteRadius(Vpbe *thee);
00215
00222 VEXTERNC double Vpbe_getSoluteXlen(Vpbe *thee);
00223

```

```

00230 VEXTERNC double Vpbe_getSoluteYlen(Vpbe *thee);
00231
00238 VEXTERNC double Vpbe_getSoluteZlen(Vpbe *thee);
00239
00246 VEXTERNC double* Vpbe_getSoluteCenter(Vpbe *thee);
00247
00254 VEXTERNC double Vpbe_getSoluteCharge(Vpbe *thee);
00255
00262 VEXTERNC double Vpbe_getSolventDiel(Vpbe *thee);
00263
00270 VEXTERNC double Vpbe_getSolventRadius(Vpbe *thee);
00271
00278 VEXTERNC double Vpbe_getXkappa(Vpbe *thee);
00279
00286 VEXTERNC double Vpbe_getDeblen(Vpbe *thee);
00287
00294 VEXTERNC double Vpbe_getZkappa2(Vpbe *thee);
00295
00302 VEXTERNC double Vpbe_getZmagic(Vpbe *thee);
00303
00304 /-----*/
00305 /* Added by Michael Grabe */
00306 /-----*/
00307
00314 VEXTERNC double Vpbe_getzmem(Vpbe *thee);
00315
00322 VEXTERNC double Vpbe_getlmem(Vpbe *thee);
00323
00330 VEXTERNC double Vpbe_getmembraneDiel(Vpbe *thee);
00331
00337 VEXTERNC double Vpbe_getmemv(Vpbe *thee);
00338
00339 /-----*/
00340
00341 #else /* if defined(VINLINE_VPBE) */
00342 # define Vpbe_getValist(thee) ((thee)->alist)
00343 # define Vpbe_getVacc(thee) ((thee)->acc)
00344 # define Vpbe_getBulkIonicStrength(thee) ((thee)->bulkIonicStrength)
00345 # define Vpbe_getTemperature(thee) ((thee)->T)
00346 # define Vpbe_getSoluteDiel(thee) ((thee)->soluteDiel)
00347 # define Vpbe_getSoluteCenter(thee) ((thee)->soluteCenter)
00348 # define Vpbe_getSoluteRadius(thee) ((thee)->soluteRadius)
00349 # define Vpbe_getSoluteXlen(thee) ((thee)->soluteXlen)
00350 # define Vpbe_getSoluteYlen(thee) ((thee)->soluteYlen)
00351 # define Vpbe_getSoluteZlen(thee) ((thee)->soluteZlen)
00352 # define Vpbe_getSoluteCharge(thee) ((thee)->soluteCharge)
00353 # define Vpbe_getSolventDiel(thee) ((thee)->solventDiel)
00354 # define Vpbe_getSolventRadius(thee) ((thee)->solventRadius)
00355 # define Vpbe_getMaxIonRadius(thee) ((thee)->maxIonRadius)
00356 # define Vpbe_getXkappa(thee) ((thee)->xkappa)
00357 # define Vpbe_getDeblen(thee) ((thee)->deblen)
00358 # define Vpbe_getZkappa2(thee) ((thee)->zkappa2)
00359 # define Vpbe_getZmagic(thee) ((thee)->zmagic)
00360
00361 /-----*/
00362 /* Added by Michael Grabe */
00363 /-----*/

```

```
00364
00365 # define Vpbe_getzmem(thee) ((thee)->z_mem)
00366 # define Vpbe_getLmem(thee) ((thee)->L)
00367 # define Vpbe_getmembraneDiel(thee) ((thee)->membraneDiel)
00368 # define Vpbe_getmemv(thee) ((thee)->V)
00369
00370 /*-----*/
00371
00372
00373 #endif /* if !defined(VINLINE_VPBE) */
00374
00375 /* /////////////////////////////////
00376 // Class Vpbe: Non-Inlineable methods (vpbe.c)
00378
0039 VEXTERNC Vpbe* Vpbe_ctor(
00400     Valist *alist,
00401     int ionNum,
00402     double *ionConc,
00403     double *ionRadii,
00404     double *ionQ,
00405     double T,
00406     double soluteDiel,
00407     double solventDiel,
00408     double solventRadius,
00409     int focusFlag,
00410     double sdens,
00411     double z_mem,
00412     double L,
00413     double membraneDiel,
00414     double V
00415 );
00416
00437 VEXTERNC int Vpbe_ctor2(
00438     Vpbe *thee,
00439     Valist *alist,
00440     int ionNum,
00441     double *ionConc,
00442     double *ionRadii,
00443     double *ionQ,
00444     double T,
00445     double soluteDiel,
00446     double solventDiel,
00447     double solventRadius,
00448     int focusFlag,
00449     double sdens,
00450     double z_mem,
00451     double L,
00452     double membraneDiel,
00453     double V
00454 );
00455
00466 VEXTERNC int Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[MAXION],
00467     double ionRadii[MAXION], double ionQ[MAXION]);
00468
00474 VEXTERNC void Vpbe_dtor(Vpbe **thee);
00475
00481 VEXTERNC void Vpbe_dtor2(Vpbe *thee);
```

```

00482
00497 VEXTERNC double Vpbe_getCoulombEnergy1(Vpbe *thee);
00498
00506 VEXTERNC unsigned long int Vpbe_memChk(Vpbe *thee);
00507
00508 #endif /* ifndef _VPBE_H_ */

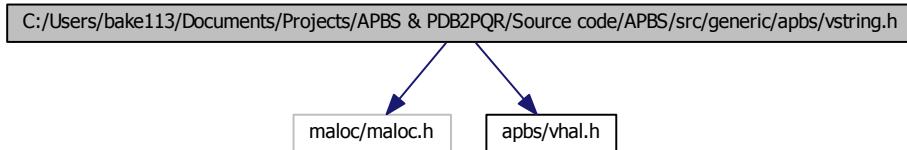
```

10.47 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vstring.h File Reference

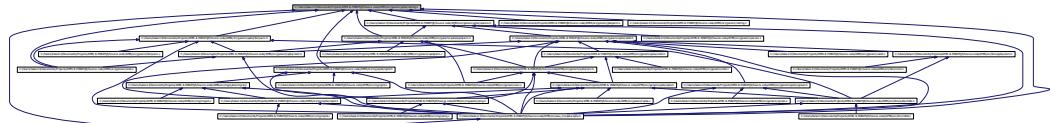
Contains declarations for class Vstring.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
```

Include dependency graph for vstring.h:



This graph shows which files directly or indirectly include this file:



Functions

- VEXTERNC int [Vstring_strcasecmp](#) (const char *s1, const char *s2)
Case-insensitive string comparison (BSD standard)
- VEXTERNC int [Vstring_isdigit](#) (const char *tok)
A modified sscanf that examines the complete string.

10.47.1 Detailed Description

Contains declarations for class Vstring.

Version

Id:

[vstring.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vstring.h](#).

10.48 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vstring.h

```

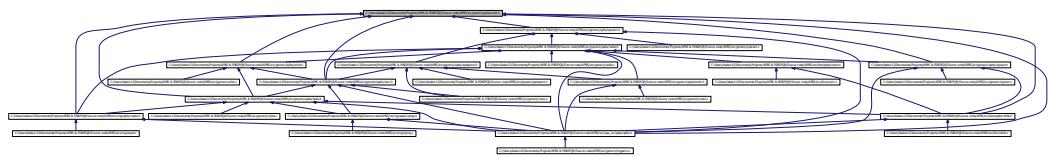
00001
00062 #ifndef _VSTRING_H_
00063 #define _VSTRING_H_
00064
00065 #include "malloc/malloc.h"
00066 #include "apbs/vhal.h"
00067
00080 VEXTERNC int Vstring_strcasecmp(const char *sl, const char *s2);
00081
00088 VEXTERNC int Vstring_isdigit(const char *tok);
00089
00090 #endif /* ifndef _VSTRING_H_ */

```

10.49 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vunit.h File Reference

Contains a collection of useful constants and conversion factors.

This graph shows which files directly or indirectly include this file:



Defines

- `#define Vunit_J_to_cal 4.1840000e+00`

Multiply by this to convert J to cal.

- #define [Vunit_cal_to_J](#) 2.3900574e-01

Multiply by this to convert cal to J.

- #define [Vunit_amu_to_kg](#) 1.6605402e-27

Multiply by this to convert amu to kg.

- #define [Vunit_kg_to_amu](#) 6.0221367e+26

Multiply by this to convert kg to amu.

- #define [Vunit_ec_to_C](#) 1.6021773e-19

Multiply by this to convert ec to C.

- #define [Vunit_C_to_ec](#) 6.2415065e+18

Multiply by this to convert C to ec.

- #define [Vunit_ec](#) 1.6021773e-19

Charge of an electron in C.

- #define [Vunit_kb](#) 1.3806581e-23

Boltzmann constant.

- #define [Vunit_Na](#) 6.0221367e+23

Avogadro's number.

- #define [Vunit_pi](#) VPI

Pi.

- #define [Vunit_eps0](#) 8.8541878e-12

Vacuum permittivity.

- #define [Vunit_esu_ec2A](#) 3.3206364e+02

e_c^2 / in ESU units => kcal/mol

- #define [Vunit_esu_kb](#) 1.9871913e-03

k_b in ESU units => kcal/mol

10.49.1 Detailed Description

Contains a collection of useful constants and conversion factors.

Author

Nathan Baker

Nathan A. Baker

Version

Id:

[vunit.h](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*
```

Definition in file [vunit.h](#).

10.50 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apbs/vunit.h

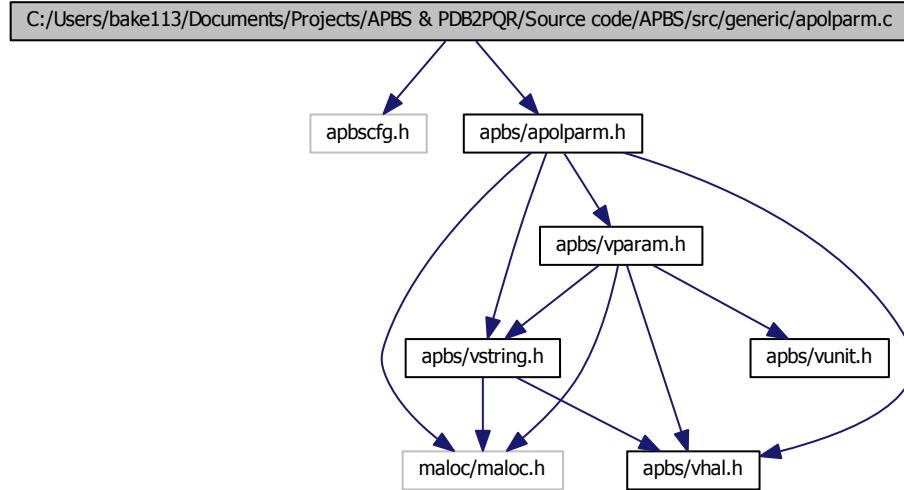
```
00001
00063 #ifndef _VUNIT_H_
00064 #define _VUNIT_H_
00065
00068 #define Vunit_J_to_cal  4.1840000e+00
00069
00072 #define Vunit_cal_to_J  2.3900574e-01
00073
00076 #define Vunit_amu_to_kg  1.6605402e-27
00077
00080 #define Vunit_kg_to_amu  6.0221367e+26
00081
00084 #define Vunit_ec_to_C  1.6021773e-19
00085
00088 #define Vunit_C_to_ec  6.2415065e+18
00089
00092 #define Vunit_ec  1.6021773e-19
00093
00096 #define Vunit_kb  1.3806581e-23
00097
00100 #define Vunit_Na  6.0221367e+23
00101
00104 #define Vunit_pi  VPI
00105
00108 #define Vunit_eps0  8.8541878e-12
00109
00112 #define Vunit_esu_ec2A  3.3206364e+02
00113
00116 #define Vunit_esu_kb          1.9871913e-03
00117
00118 #endif /* ifndef _VUNIT_H_ */
```

10.51 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apolparm.c File Reference

Class APOLparm methods.

```
#include "apbscfg.h"
#include "apbs/apolparm.h"
```

Include dependency graph for apolparm.c:



Functions

- VPUBLIC [APOLparm](#) * [APOLparm_ctor](#) ()

Construct APOLparm.
- VPUBLIC Vrc_Codes [APOLparm_ctor2](#) ([APOLparm](#) *thee)

FORTRAN stub to construct APOLparm.
- VPUBLIC void [APOLparm_copy](#) ([APOLparm](#) *thee, [APOLparm](#) *source)

Copy target object into thee.
- VPUBLIC void [APOLparm_dtor](#) ([APOLparm](#) **thee)

Object destructor.
- VPUBLIC void [APOLparm_dtor2](#) ([APOLparm](#) *thee)

FORTRAN stub for object destructor.
- VPUBLIC Vrc_Codes [APOLparm_check](#) ([APOLparm](#) *thee)

Consistency check for parameter values stored in object.
- VPRIVATE Vrc_Codes [APOLparm_parseGRID](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseMOL](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseSRFM](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseSRAD](#) ([APOLparm](#) *thee, Vio *sock)

- VPRIVATE Vrc_Codes **APOLparm_parseSWIN** ([APOLparm](#) *thee, [Vio](#) *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseTEMP** ([APOLparm](#) *thee, [Vio](#) *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseGAMMA** ([APOLparm](#) *thee, [Vio](#) *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseCALCENERGY** ([APOLparm](#) *thee, [Vio](#) *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseCALCFORCE** ([APOLparm](#) *thee, [Vio](#) *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseBCONC** ([APOLparm](#) *thee, [Vio](#) *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseSDENS** ([APOLparm](#) *thee, [Vio](#) *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseDPOS** ([APOLparm](#) *thee, [Vio](#) *sock)
- VPRIVATE Vrc_Codes **APOLparm_parsePRESS** ([APOLparm](#) *thee, [Vio](#) *sock)
- VPUBLIC Vrc_Codes **APOLparm_parseToken** ([APOLparm](#) *thee, char tok[VMAX_-BUFSIZE], [Vio](#) *sock)

Parse an MG keyword from an input file.

10.51.1 Detailed Description

Class APOLparm methods.

Author

David Gohara

Version

Id:

[apolparm.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
```

```

*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [apolparm.c](#).

10.52 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/apolparm.c

```

00001
00058 #include "apbscfg.h"
00059 #include "apbs/apolparm.h"
00060
00061 VEMBED(rcsid="$Id: apolparm.c 1667 2011-12-02 23:22:02Z pcellis $")
00062
00063 #if !defined(VINLINE_MGPARM)
00064
00065 #endif /* if !defined(VINLINE_MGPARM) */
00066
00067 VPUBLIC APOLparm* APOLparm_ctor() {
00068

```

```

00069     /* Set up the structure */
00070     APOLparm *thee = VNULL;
00071     thee = Vmem_malloc(VNULL, 1, sizeof(APOLparm));
00072     VASSERT( thee != VNULL );
00073     VASSERT( APOLparm_ctor2(thee) == VRC_SUCCESS );
00074
00075     return thee;
00076 }
00077
00078 VPUBLIC Vrc_Codes APOLparm_ctor2(APOLparm *thee) {
00079
00080     int i;
00081
00082     if (thee == VNULL) return VRC_FAILURE;
00083
00084     thee->parsed = 0;
00085
00086     thee->setgrid = 0;
00087     thee->setmolid = 0;
00088     thee->setbconc = 0;
00089     thee->setsdens = 0;
00090     thee->setdpos = 0;
00091     thee->setpress = 0;
00092     thee->setsrfm = 0;
00093     thee->setsrad = 0;
00094     thee->setswin = 0;
00095
00096     thee->settemp = 0;
00097     thee->setgamma = 0;
00098
00099     thee->setwat = 0;
00100
00101     thee->sav = 0.0;
00102     thee->sasa = 0.0;
00103     thee->wcaEnergy = 0.0;
00104
00105     for(i=0;i<3;i++) thee->totForce[i] = 0.0;
00106
00107     return VRC_SUCCESS;
00108 }
00109
00110 VPUBLIC void APOLparm_copy(
00111         APOLparm *thee,
00112         APOLparm *source
00113     ) {
00114
00115     int i;
00116
00117     thee->parsed = source->parsed;
00118
00119     for (i=0; i<3; i++) thee->grid[i] = source->grid[i];
00120     thee->setgrid = source->setgrid;
00121
00122     thee->molid = source->molid;
00123     thee->setmolid = source->setmolid;
00124
00125     thee->bconc = source->bconc ;

```

```

00126 thee->setbconc= source->setbconc ;
00127
00128 thee->sdens = source->sdens ;
00129 thee->setsdens= source->setsdens ;
00130
00131 thee->dpos = source->dpos ;
00132 thee->setdpos= source->setdpos ;
00133
00134 thee->press = source->press ;
00135 thee->setpress = source->setpress ;
00136
00137 thee->srfm = source->srfm ;
00138 thee->setsrfm = source->setsrfm ;
00139
00140 thee->srad = source->srad ;
00141 thee->setsrad = source->setsrad ;
00142
00143 thee->swin = source->swin ;
00144 thee->setswin = source->setswin ;
00145
00146 thee->temp = source->temp ;
00147 thee->settemp = source->settemp ;
00148
00149 thee->gamma = source->gamma ;
00150 thee->setgamma = source->setgamma ;
00151
00152 thee->calcenergy = source->calcenergy ;
00153 thee->setcalcenergy = source->setcalcenergy ;
00154
00155 thee->calccforce = source->calccforce ;
00156 thee->setcalccforce = source->setcalccforce ;
00157
00158 thee->setwat = source->setwat ;
00159
00160 thee->sav = source->sav;
00161 thee->sasa = source->sasa;
00162 thee->wcaEnergy = source->wcaEnergy;
00163
00164 for(i=0;i<3;i++) thee->totForce[i] = source->totForce[i];
00165
00166 return;
00167 }
00168
00169 VPUBLIC void APOLparm_dtor(APOLparm **thee) {
00170     if ((*thee) != VNULL) {
00171         APOLparm_dtor2(*thee);
00172         Vmem_free(VNULL, 1, sizeof(APOLparm), (void **)thee);
00173         (*thee) = VNULL;
00174     }
00175
00176     return;
00177 }
00178
00179 VPUBLIC void APOLparm_dtor2(APOLparm *thee) { ; }
00180
00181 VPUBLIC Vrc_Codes APOLparm_check(APOLparm *thee) {
00182

```

```

00183     Vrc_Codes rc;
00184     rc = VRC_SUCCESS;
00185
00186     if (!thee->parsed) {
00187         Vnm_print(2, "APOLparm_check: not filled!\n");
00188         return VRC_FAILURE;
00189     }
00190     if (!thee->setgrid) {
00191         Vnm_print(2, "APOLparm_check: grid not set!\n");
00192         rc = VRC_FAILURE;
00193     }
00194     if (!thee->setmolid) {
00195         Vnm_print(2, "APOLparm_check: molid not set!\n");
00196         rc = VRC_FAILURE;
00197     }
00198     if (!thee->setbconc) {
00199         Vnm_print(2, "APOLparm_check: bconc not set!\n");
00200         rc = VRC_FAILURE;
00201     }
00202     if (!thee->setsdens) {
00203         Vnm_print(2, "APOLparm_check: sdens not set!\n");
00204         rc = VRC_FAILURE;
00205     }
00206     if (!thee->setdpos) {
00207         Vnm_print(2, "APOLparm_check: dpos not set!\n");
00208         rc = VRC_FAILURE;
00209     }
00210     if (!thee->setpress) {
00211         Vnm_print(2, "APOLparm_check: press not set!\n");
00212         rc = VRC_FAILURE;
00213     }
00214     if (!thee->setsrfm) {
00215         Vnm_print(2, "APOLparm_check: srfm not set!\n");
00216         rc = VRC_FAILURE;
00217     }
00218     if (!thee->setsrad) {
00219         Vnm_print(2, "APOLparm_check: srad not set!\n");
00220         rc = VRC_FAILURE;
00221     }
00222     if (!thee->setswin) {
00223         Vnm_print(2, "APOLparm_check: swin not set!\n");
00224         rc = VRC_FAILURE;
00225     }
00226     if (!thee->settemp) {
00227         Vnm_print(2, "APOLparm_check: temp not set!\n");
00228         rc = VRC_FAILURE;
00229     }
00230 }
00231 if (!thee->setgamma) {
00232     Vnm_print(2, "APOLparm_check: gamma not set!\n");
00233     rc = VRC_FAILURE;
00234 }
00235 return rc;
00236
00237 }
00238
00239 VPRIVATE Vrc_Codes APOLparm_parseGRID(APOLparm *thee, Vio *sock) {

```

```

00240
00241     char tok[VMAX_BUFSIZE];
00242     double tf;
00243
00244     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00245     if (sscanf(tok, "%lf", &tf) == 0) {
00246         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GRID \
00247 keyword!\n", tok);
00248         return VRC_WARNING;
00249     } else thee->grid[0] = tf;
00250     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00251     if (sscanf(tok, "%lf", &tf) == 0) {
00252         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GRID \
00253 keyword!\n", tok);
00254         return VRC_WARNING;
00255     } else thee->grid[1] = tf;
00256     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00257     if (sscanf(tok, "%lf", &tf) == 0) {
00258         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GRID \
00259 keyword!\n", tok);
00260         return VRC_WARNING;
00261     } else thee->grid[2] = tf;
00262     thee->setgrid = 1;
00263     return VRC_SUCCESS;
00264
00265 VERROR1:
00266     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00267     return VRC_WARNING;
00268 }
00269
00270 VPRIVATE Vrc_Codes APOLparm_parseMOL(APOLparm *thee, Vio *sock) {
00271     int ti;
00272     char tok[VMAX_BUFSIZE];
00273
00274     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00275     if (sscanf(tok, "%d", &ti) == 0) {
00276         Vnm_print(2, "NOsh: Read non-int (%s) while parsing MOL \
00277 keyword!\n", tok);
00278         return VRC_WARNING;
00279     }
00280     thee->molid = ti;
00281     thee->setmolid = 1;
00282     return VRC_SUCCESS;
00283
00284 VERROR1:
00285     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00286     return VRC_WARNING;
00287 }
00288
00289 VPRIVATE Vrc_Codes APOLparm_parseSRFM(APOLparm *thee, Vio *sock) {
00290     char tok[VMAX_BUFSIZE];
00291
00292     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00293
00294     if (Vstring_strcasecmp(tok, "sacc") == 0) {
00295         thee->srfm = VSM_MOL;
00296         thee->setsrfm = 1;

```

```

00297         return VRC_SUCCESS;
00298     } else {
00299         printf("parseAPOL: Unrecognized keyword (%s) when parsing srfm!\n", tok)
00300 ;
00301         printf("parseAPOL: Accepted values for srfm = sacc\n");
00302         return VRC_WARNING;
00303     }
00304     return VRC_FAILURE;
00305
00306 VERROR1:
00307     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00308     return VRC_WARNING;
00309 }
00310
00311 VPRIVATE Vrc_Codes APOLparm_parseSRAD(APOLparm *thee, Vio *sock) {
00312     char tok[VMAX_BUFSIZE];
00313     double tf;
00314
00315     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00316     if (sscanf(tok, "%lf", &tf) == 0) {
00317         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SRAD \
00318 keyword!\n", tok);
00319         return VRC_WARNING;
00320     }
00321     thee->srad = tf;
00322     thee->setsrad = 1;
00323     return VRC_SUCCESS;
00324
00325 VERROR1:
00326     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00327     return VRC_WARNING;
00328 }
00329
00330 VPRIVATE Vrc_Codes APOLparm_parseSWIN(APOLparm *thee, Vio *sock) {
00331     char tok[VMAX_BUFSIZE];
00332     double tf;
00333
00334     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00335     if (sscanf(tok, "%lf", &tf) == 0) {
00336         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SWIN \
00337 keyword!\n", tok);
00338         return VRC_WARNING;
00339     }
00340     thee->swin = tf;
00341     thee->setswin = 1;
00342     return VRC_SUCCESS;
00343
00344 VERROR1:
00345     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00346     return VRC_WARNING;
00347 }
00348
00349 VPRIVATE Vrc_Codes APOLparm_parseTEMP(APOLparm *thee, Vio *sock) {
00350     char tok[VMAX_BUFSIZE];
00351     double tf;
00352

```

```

00353     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00354     if (sscanf(tok, "%lf", &tf) == 0) {
00355         Vnm_print(2, "NOsh: Read non-float (%s) while parsing TEMP \
00356 keyword!\n", tok);
00357         return VRC_WARNING;
00358     }
00359     thee->temp = tf;
00360     thee->settemp = 1;
00361     return VRC_SUCCESS;
00362
00363 VERROR1:
00364     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00365     return VRC_WARNING;
00366 }
00367
00368 VPRIVATE Vrc_Codes APOLparm_parseGAMMA(APOLparm *thee, Vio *sock) {
00369     char tok[VMAX_BUFSIZE];
00370     double tf;
00371
00372     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00373     if (sscanf(tok, "%lf", &tf) == 0) {
00374         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GAMMA \
00375 keyword!\n", tok);
00376         return VRC_WARNING;
00377     }
00378     thee->gamma = tf;
00379     thee->setgamma = 1;
00380     return VRC_SUCCESS;
00381
00382 VERROR1:
00383     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00384     return VRC_WARNING;
00385 }
00386
00387 VPRIVATE Vrc_Codes APOLparm_parseCALCENERGY(APOLparm *thee, Vio *sock) {
00388     char tok[VMAX_BUFSIZE];
00389     int ti;
00390
00391     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00392     /* Parse number */
00393     if (sscanf(tok, "%d", &ti) == 1) {
00394         thee->calcenergy = ti;
00395         thee->setcalcenergy = 1;
00396
00397         Vnm_print(2, "parseAPOL: Warning -- parsed deprecated \"calcenergy \
00398 %d\" statement.\n", ti);
00399         Vnm_print(2, "parseAPOL: Please use \"calcenergy \"");
00400         switch (thee->calcenergy) {
00401             case ACE_NO:
00402                 Vnm_print(2, "no");
00403                 break;
00404             case ACE_TOTAL:
00405                 Vnm_print(2, "total");
00406                 break;
00407             case ACE_COMPS:
00408                 Vnm_print(2, "comps");
00409                 break;

```

```

00410         default:
00411             Vnm_print(2, "UNKNOWN");
00412             break;
00413         }
00414         Vnm_print(2, "\ instead.\n");
00415         return VRC_SUCCESS;
00416     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00417         thee->calcenergy = ACE_NO;
00418         thee->setcalcenergy = 1;
00419         return VRC_SUCCESS;
00420     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00421         thee->calcenergy = ACE_TOTAL;
00422         thee->setcalcenergy = 1;
00423         return VRC_SUCCESS;
00424     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00425         thee->calcenergy = ACE_COMPS;
00426         thee->setcalcenergy = 1;
00427         return VRC_SUCCESS;
00428     } else {
00429         Vnm_print(2, "NOsh: Unrecognized parameter (%s) while parsing \
00430 calcenergy!\n", tok);
00431         return VRC_WARNING;
00432     }
00433     return VRC_FAILURE;
00434 }
00435 VERROR1:
00436     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00437     return VRC_WARNING;
00438 }
00439
00440 VPRIvATE Vrc_Codes APOLparm_parseCALCFORCE(APOLparm *thee, Vio *sock) {
00441     char tok[VMAX_BUFSIZE];
00442     int ti;
00443
00444     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00445     /* Parse number */
00446     if (sscanf(tok, "%d", &ti) == 1) {
00447         thee->calcforce = ti;
00448         thee->setcalcforce = 1;
00449
00450         Vnm_print(2, "parseAPOL: Warning -- parsed deprecated \"calcforce \
00451 %d\" statement.\n", ti);
00452         Vnm_print(2, "parseAPOL: Please use \"calcforce \"");
00453         switch (thee->calcenergy) {
00454             case ACF_NO:
00455                 Vnm_print(2, "no");
00456                 break;
00457             case ACF_TOTAL:
00458                 Vnm_print(2, "total");
00459                 break;
00460             case ACF_COMPS:
00461                 Vnm_print(2, "comps");
00462                 break;
00463             default:
00464                 Vnm_print(2, "UNKNOWN");
00465                 break;
00466         }

```

```

00467      Vnm_print(2, "\" instead.\n");
00468      return VRC_SUCCESS;
00469  } else if (Vstring_strcasecmp(tok, "no") == 0) {
00470      thee->calcforce = ACF_NO;
00471      thee->setcalcforce = 1;
00472      return VRC_SUCCESS;
00473  } else if (Vstring_strcasecmp(tok, "total") == 0) {
00474      thee->calcforce = ACF_TOTAL;
00475      thee->setcalcforce = 1;
00476      return VRC_SUCCESS;
00477  } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00478      thee->calcforce = ACF_COMPS;
00479      thee->setcalcforce = 1;
00480      return VRC_SUCCESS;
00481  } else {
00482      Vnm_print(2, "NOsh: Unrecognized parameter (%s) while parsing \
00483 calcforce!\n", tok);
00484      return VRC_WARNING;
00485  }
00486  return VRC_FAILURE;
00487
00488 VERROR1:
00489     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00490     return VRC_WARNING;
00491 }
00492
00493 VPRIVATE Vrc_Codes APOLparm_parseBCONC(APOLparm *thee, Vio *sock) {
00494     char tok[VMAX_BUFSIZE];
00495     double tf;
00496
00497     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00498     if (sscanf(tok, "%lf", &tf) == 0) {
00499         Vnm_print(2, "NOsh: Read non-float (%s) while parsing BCONC \
00500 keyword!\n", tok);
00501         return VRC_WARNING;
00502     }
00503     thee->bconc = tf;
00504     thee->setbconc = 1;
00505     return VRC_SUCCESS;
00506
00507 VERROR1:
00508     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00509     return VRC_WARNING;
00510 }
00511
00512 VPRIVATE Vrc_Codes APOLparm_parseSDENS(APOLparm *thee, Vio *sock) {
00513     char tok[VMAX_BUFSIZE];
00514     double tf;
00515
00516     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00517     if (sscanf(tok, "%lf", &tf) == 0) {
00518         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SDENS \
00519 keyword!\n", tok);
00520         return VRC_WARNING;
00521     }
00522     thee->sdens = tf;
00523     thee->setsdens = 1;

```

```

00524     return VRC_SUCCESS;
00525
00526 VERROR1:
00527     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00528     return VRC_WARNING;
00529 }
00530
00531 VPRIIVATE Vrc_Codes APOLparm_parseDPOS(APOLparm *thee, Vio *sock) {
00532     char tok[VMAX_BUFSIZE];
00533     double tf;
00534
00535     VJMPERR1(Vic_scantok(sock, "%s", tok) == 1);
00536     if (sscanf(tok, "%lf", &tf) == 0) {
00537         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SDENS \
00538 keyword!\n", tok);
00539         return VRC_WARNING;
00540     }
00541     thee->dpos = tf;
00542     thee->setdpos = 1;
00543
00544     if (thee->dpos < 0.001) {
00545         Vnm_print(1, "\nWARNING WARNING WARNING WARNING WARNING\n");
00546         Vnm_print(1, "NOsh: dpos is set to a very small value.\n");
00547         Vnm_print(1, "NOsh: If you are not using a PQR file, you can \
00548 safely ignore this message.\n");
00549         Vnm_print(1, "NOsh: Otherwise please choose a value greater than \
00550 or equal to 0.001.\n\n");
00551     }
00552
00553     return VRC_SUCCESS;
00554
00555 VERROR1:
00556     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00557     return VRC_WARNING;
00558 }
00559
00560 VPRIIVATE Vrc_Codes APOLparm_parsePRESS(APOLparm *thee, Vio *sock) {
00561     char tok[VMAX_BUFSIZE];
00562     double tf;
00563
00564     VJMPERR1(Vic_scantok(sock, "%s", tok) == 1);
00565     if (sscanf(tok, "%lf", &tf) == 0) {
00566         Vnm_print(2, "NOsh: Read non-float (%s) while parsing PRESS \
00567 keyword!\n", tok);
00568         return VRC_WARNING;
00569     }
00570     thee->press = tf;
00571     thee->setpress = 1;
00572     return VRC_SUCCESS;
00573
00574 VERROR1:
00575     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00576     return VRC_WARNING;
00577 }
00578
00579 VPUBLIC Vrc_Codes APOLparm_parseToken(APOLparm *thee, char tok[VMAX_BUFSIZE],
00580     Vio *sock) {

```

```

00581
00582     if (thee == VNULL) {
00583         Vnm_print(2, "parseAPOL: got NULL thee!\n");
00584         return VRC_WARNING;
00585     }
00586
00587     if (sock == VNULL) {
00588         Vnm_print(2, "parseAPOL: got NULL socket!\n");
00589         return VRC_WARNING;
00590     }
00591
00592     if (Vstring_strcasecmp(tok, "mol") == 0) {
00593         return APOLparm_parseMOL(thee, sock);
00594     } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00595         return APOLparm_parseGRID(thee, sock);
00596     } else if (Vstring_strcasecmp(tok, "dime") == 0) {
00597         Vnm_print(2, "APOLparm_parseToken: The DIME and GLEN keywords for APOLAR have
00598         been replaced with GRID.\n");
00599         Vnm_print(2, "APOLparm_parseToken: Please see the APBS User Guide for more inf
00600         ormation.\n");
00601         return VRC_WARNING;
00602     } else if (Vstring_strcasecmp(tok, "glen") == 0) {
00603         Vnm_print(2, "APOLparm_parseToken: The DIME and GLEN keywords for APOLAR have
00604         been replaced with GRID.\n");
00605         Vnm_print(2, "APOLparm_parseToken: Please see the APBS User Guide for more inf
00606         ormation.\n");
00607         return VRC_WARNING;
00608     } else if (Vstring_strcasecmp(tok, "bconc") == 0) {
00609         return APOLparm_parseBCONC(thee, sock);
00610     } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
00611         return APOLparm_parseSDENS(thee, sock);
00612     } else if (Vstring_strcasecmp(tok, "dpos") == 0) {
00613         return APOLparm_parseDPOS(thee, sock);
00614     } else if (Vstring_strcasecmp(tok, "srfm") == 0) {
00615         return APOLparm_parseSRFM(thee, sock);
00616     } else if (Vstring_strcasecmp(tok, "srad") == 0) {
00617         return APOLparm_parseSRAD(thee, sock);
00618     } else if (Vstring_strcasecmp(tok, "swin") == 0) {
00619         return APOLparm_parseSWIN(thee, sock);
00620     } else if (Vstring_strcasecmp(tok, "temp") == 0) {
00621         return APOLparm_parseTEMP(thee, sock);
00622     } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
00623         return APOLparm_parseGAMMA(thee, sock);
00624     } else if (Vstring_strcasecmp(tok, "press") == 0) {
00625         return APOLparm_parsePRESS(thee, sock);
00626     } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
00627         return APOLparm_parseCALCENERGY(thee, sock);
00628     } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
00629         return APOLparm_parseCALCFORCE(thee, sock);
00630     } else {
00631         Vnm_print(2, "parseAPOL: Unrecognized keyword (%s)!\n", tok);
00632         return VRC_WARNING;
00633     }
00634
00635
00636
00637     return VRC_FAILURE;
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01778
01779
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01878
01879
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01978
01979
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288

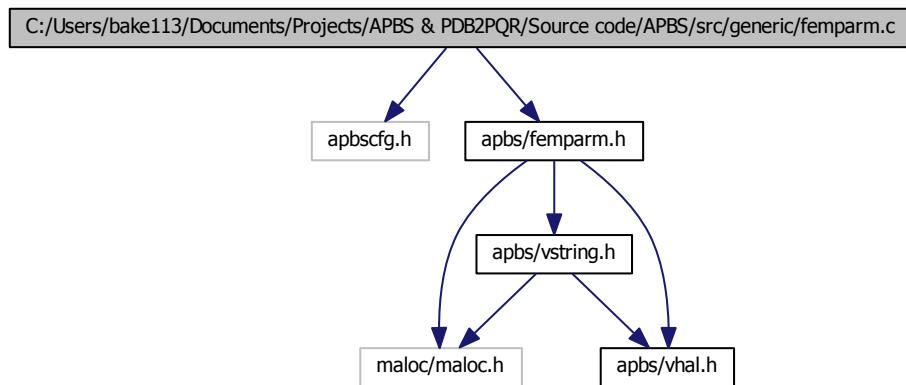
```

```
00634 }
```

10.53 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/femparm.c File Reference

Class FEMparm methods.

```
#include "apbscfg.h"  
#include "apbs/femparm.h"  
Include dependency graph for femparm.c:
```



Functions

- VPUBLIC FEMparm * FEMparm_ctor (FEMparm_CalcType type)
Construct FEMparm.
- VPUBLIC int FEMparm_ctor2 (FEMparm *thee, FEMparm_CalcType type)
FORTRAN stub to construct FEMparm.
- VPUBLIC void FEMparm_copy (FEMparm *thee, FEMparm *source)
Copy target object into thee.
- VPUBLIC void FEMparm_dtor (FEMparm **thee)
Object destructor.
- VPUBLIC void FEMparm_dtor2 (FEMparm *thee)

FORTRAN stub for object destructor.

- VPUBLIC int **FEMparm_check** (**FEMparm** *thee)

Consistency check for parameter values stored in object.
- VPRIVATE Vrc_Codes **FEMparm_parseDOMAINLENGTH** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseETOL** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseEKEY** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseAKEYPRE** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseAKEYSOLVE** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseTARGETNUM** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseTARGETRES** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseMAXSOLVE** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseMAXVERT** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseUSEMESH** (**FEMparm** *thee, Vio *sock)
- VPUBLIC Vrc_Codes **FEMparm_parseToken** (**FEMparm** *thee, char tok[VMAX_BUFSIZE], Vio *sock)

Parse an MG keyword from an input file.

10.53.1 Detailed Description

Class FEMparm methods.

Author

Nathan Baker

Version

Id:

[femparm.c](#) 1667 2011-12-02 23:22:02Z pcells

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*
```

```
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [femparm.c](#).

10.54 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/femparm.c

```
00001
00058 #include "apbscfg.h"
00059 #include "apbs/femparm.h"
00060
```

```

00061 VEMBED(rcsid="$Id: femparm.c 1667 2011-12-02 23:22:02Z pcellis $")
00062
00063 #if !defined(VINLINE_MGPARM)
00064
00065 #endif /* if !defined(VINLINE_MGPARM) */
00066
00067 VPUBLIC FEMparm* FEMparm_ctor(FEMparm_CalcType type) {
00068
00069     /* Set up the structure */
00070     FEMparm *thee = VNULL;
00071     thee = Vmem_malloc(VNULL, 1, sizeof(FEMparm));
00072     VASSERT( thee != VNULL );
00073     VASSERT( FEMparm_ctor2(thee, type) );
00074
00075     return thee;
00076 }
00077
00078 VPUBLIC int FEMparm_ctor2(FEMparm *thee, FEMparm_CalcType type) {
00079
00080     if (thee == VNULL) return 0;
00081
00082     thee->parsed = 0;
00083     thee->type = type;
00084     thee->settype = 1;
00085
00086     thee->setglen = 0;
00087     thee->setetol = 0;
00088     thee->setekey = 0;
00089     thee->setakeyPRE = 0;
00090     thee->setakeySOLVE = 0;
00091     thee->settargtNum = 0;
00092     thee->settargtRes = 0;
00093     thee->setmaxsolve = 0;
00094     thee->setmaxvert = 0;
00095     thee->useMesh = 0;
00096
00097     return 1;
00098 }
00099
00100 VPUBLIC void FEMparm_copy(
00101         FEMparm *thee,
00102         FEMparm *source
00103     ) {
00104
00105     int i;
00106
00107     thee->parsed = source->parsed;
00108     thee->type = source->type;
00109     thee->settype = source->settype;
00110     for (i=0; i<3; i++) thee->glen[i] = source->glen[i];
00111     thee->setglen = source->setglen;
00112     thee->etol = source->etol;
00113     thee->setetol = source->setetol;
00114     thee->ekey = source->ekey;
00115     thee->setekey = source->setekey;
00116     thee->akeyPRE = source->akeyPRE;
00117     thee->setakeyPRE = source->setakeyPRE;

```

```

00118     thee->akeySOLVE = source->akeySOLVE;
00119     thee->setakeySOLVE = source->setakeySOLVE;
00120     thee->targetNum = source->targetNum;
00121     thee->settargetNum = source->settargetNum;
00122     thee->targetRes = source->targetRes;
00123     thee->settargetRes = source->settargetRes;
00124     thee->maxsolve = source->maxsolve;
00125     thee->setmaxsolve = source->setmaxsolve;
00126     thee->maxvert = source->maxvert;
00127     thee->setmaxvert = source->setmaxvert;
00128     thee->pkey = source->pkey;
00129     thee->useMesh = source->useMesh;
00130     thee->meshID = source->meshID;
00131 }
00132
00133 VPUBLIC void FEMPARM_dtor(FEMPARM **thee) {
00134     if ((*thee) != VNULL) {
00135         FEMPARM_dtor2(*thee);
00136         Vmem_free(VNULL, 1, sizeof(FEMPARM), (void **)thee);
00137         (*thee) = VNULL;
00138     }
00139 }
00140
00141 VPUBLIC void FEMPARM_dtor2(FEMPARM *thee) { ; }
00142
00143 VPUBLIC int FEMPARM_check(FEMPARM *thee) {
00144
00145     int rc;
00146     rc = 1;
00147
00148     if (!thee->parsed) {
00149         Vnm_print(2, "FEMPARM_check: not filled!\n");
00150         return 0;
00151     }
00152     if (!thee->settype) {
00153         Vnm_print(2, "FEMPARM_check: type not set!\n");
00154         rc = 0;
00155     }
00156     if (!thee->setglen) {
00157         Vnm_print(2, "FEMPARM_check: glen not set!\n");
00158         rc = 0;
00159     }
00160     if (!thee->setetol) {
00161         Vnm_print(2, "FEMPARM_check: etol not set!\n");
00162         rc = 0;
00163     }
00164     if (!thee->setekey) {
00165         Vnm_print(2, "FEMPARM_check: ekey not set!\n");
00166         rc = 0;
00167     }
00168     if (!thee->setakeyPRE) {
00169         Vnm_print(2, "FEMPARM_check: akeyPRE not set!\n");
00170         rc = 0;
00171     }
00172     if (!thee->setakeySOLVE) {
00173         Vnm_print(2, "FEMPARM_check: akeySOLVE not set!\n");
00174         rc = 0;

```

```

00175     }
00176     if (!thee->settargetNum) {
00177         Vnm_print(2, "FEMparm_check: targetNum not set!\n");
00178         rc = 0;
00179     }
00180     if (!thee->settargetRes) {
00181         Vnm_print(2, "FEMparm_check: targetRes not set!\n");
00182         rc = 0;
00183     }
00184     if (!thee->setmaxsolve) {
00185         Vnm_print(2, "FEMparm_check: maxsolve not set!\n");
00186         rc = 0;
00187     }
00188     if (!thee->setmaxvert) {
00189         Vnm_print(2, "FEMparm_check: maxvert not set!\n");
00190         rc = 0;
00191     }
00192
00193     return rc;
00194 }
00195
00196 VPRIVATE Vrc_Codes FEMparm_parseDOMAINLENGTH(FEMparm *thee, Vio *sock) {
00197
00198     int i;
00199     double tf;
00200     char tok[VMAX_BUFSIZE];
00201
00202     for (i=0; i<3; i++) {
00203         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00204         if (sscanf(tok, "%lf", &tf) == 0) {
00205             Vnm_print(2, "parseFE: Read non-double (%s) while parsing \
00206 DOMAINLENGTH keyword!\n", tok);
00207             return VRC_FAILURE;
00208         }
00209         thee->glen[i] = tf;
00210     }
00211     thee->setglen = 1;
00212     return VRC_SUCCESS;
00213 VERROR1:
00214     Vnm_print(2, "parseFE: ran out of tokens!\n");
00215     return VRC_FAILURE;
00216
00217 }
00218
00219 VPRIVATE Vrc_Codes FEMparm_parseETOL(FEMparm *thee, Vio *sock) {
00220
00221     double tf;
00222     char tok[VMAX_BUFSIZE];
00223
00224     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00225     if (sscanf(tok, "%lf", &tf) == 0) {
00226         Vnm_print(2, "parseFE: Read non-double (%s) while parsing \
00227 ETOL keyword!\n", tok);
00228         return VRC_FAILURE;
00229     }
00230     thee->etol = tf;
00231     thee->setetol = 1;

```

```

00232     return VRC_SUCCESS;
00233 VERROR1:
00234     Vnm_print(2, "parseFE: ran out of tokens!\n");
00235     return VRC_FAILURE;
00236
00237
00238 }
00239
00240 VPRIVATE Vrc_Codes FEmparm_parseEKEY(FEmparm *thee, Vio *sock) {
00241
00242     char tok[VMAX_BUFSIZE];
00243     Vrc_Codes vrc = VRC_FAILURE;
00244
00245     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00246     if (Vstring_strcasecmp(tok, "simp") == 0) {
00247         thee->ekey = FET_SIMP;
00248         thee->setekey = 1;
00249         vrc = VRC_SUCCESS;
00250     } else if (Vstring_strcasecmp(tok, "glob") == 0) {
00251         thee->ekey = FET_GLOB;
00252         thee->setekey = 1;
00253         vrc = VRC_SUCCESS;
00254     } else if (Vstring_strcasecmp(tok, "frac") == 0) {
00255         thee->ekey = FET_FRAC;
00256         thee->setekey = 1;
00257         vrc = VRC_SUCCESS;
00258     } else {
00259         Vnm_print(2, "parseFE: undefined value (%s) for ekey!\n", tok);
00260         vrc = VRC_FAILURE;
00261     }
00262
00263     return vrc;
00264 VERROR1:
00265     Vnm_print(2, "parseFE: ran out of tokens!\n");
00266     return VRC_FAILURE;
00267
00268 }
00269
00270 VPRIVATE Vrc_Codes FEmparm_parseAKEYPRE(FEmparm *thee, Vio *sock) {
00271
00272     char tok[VMAX_BUFSIZE];
00273     Vrc_Codes vrc = VRC_FAILURE;
00274
00275     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00276     if (Vstring_strcasecmp(tok, "unif") == 0) {
00277         thee->akeyPRE = FRT_UNIF;
00278         thee->setakeyPRE = 1;
00279         vrc = VRC_SUCCESS;
00280     } else if (Vstring_strcasecmp(tok, "geom") == 0) {
00281         thee->akeyPRE = FRT_GEOM;
00282         thee->setakeyPRE = 1;
00283         vrc = VRC_SUCCESS;
00284     } else {
00285         Vnm_print(2, "parseFE: undefined value (%s) for akeyPRE!\n", tok);
00286         vrc = VRC_FAILURE;
00287     }
00288 }
```

```

00289     return vrc;
00290
00291 VERROR1:
00292     Vnm_print(2, "parseFE: ran out of tokens!\n");
00293     return VRC_FAILURE;
00294
00295 }
00296
00297 VPRIvate Vrc_Codes FEMparm_parseAKEYSOLVE(FEMparm *thee, Vio *sock) {
00298
00299     char tok[VMAX_BUFSIZE];
00300     Vrc_Codes vrc = VRC_FAILURE;
00301
00302     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00303     if (Vstring_strcasecmp(tok, "resi") == 0) {
00304         thee->akeySOLVE = FRT_RESI;
00305         thee->setakeySOLVE = 1;
00306         vrc = VRC_SUCCESS;
00307     } else if (Vstring_strcasecmp(tok, "dual") == 0) {
00308         thee->akeySOLVE = FRT_DUAL;
00309         thee->setakeySOLVE = 1;
00310         vrc = VRC_SUCCESS;
00311     } else if (Vstring_strcasecmp(tok, "loca") == 0) {
00312         thee->akeySOLVE = FRT_LOCA;
00313         thee->setakeySOLVE = 1;
00314         vrc = VRC_SUCCESS;
00315     } else {
00316         Vnm_print(2, "parseFE: undefined value (%s) for akeyPRE!\n", tok);
00317         vrc = VRC_FAILURE;
00318     }
00319
00320     return vrc;
00321 VERROR1:
00322     Vnm_print(2, "parseFE: ran out of tokens!\n");
00323     return VRC_SUCCESS;
00324
00325 }
00326
00327 VPRIvate Vrc_Codes FEMparm_parseTARGETNUM(FEMparm *thee, Vio *sock) {
00328
00329     char tok[VMAX_BUFSIZE];
00330     int ti;
00331
00332     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00333     if (sscanf(tok, "%d", &ti) == 0) {
00334         Vnm_print(2, "parseFE: read non-int (%s) for targetNum!\n", tok);
00335         return VRC_FAILURE;
00336     }
00337     thee->targetNum = ti;
00338     thee->settargetNum = 1;
00339     return VRC_SUCCESS;
00340 VERROR1:
00341     Vnm_print(2, "parseFE: ran out of tokens!\n");
00342     return VRC_FAILURE;
00343
00344 }
00345

```

```

00346 VPRIVATE Vrc_Codes FEmparm_parseTARGETRES(FEmparm *thee, Vio *sock) {
00347
00348     char tok[VMAX_BUFSIZE];
00349     double tf;
00350
00351     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00352     if (sscanf(tok, "%lf", &tf) == 0) {
00353         Vnm_print(2, "parseFE:  read non-double (%s) for targetNum!\n",
00354                 tok);
00355         return VRC_FAILURE;
00356     }
00357     thee->targetRes = tf;
00358     thee->settargetRes = 1;
00359     return VRC_SUCCESS;
00360 VERROR1:
00361     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00362     return VRC_FAILURE;
00363 }
00364 }
00365
00366 VPRIVATE Vrc_Codes FEmparm_parseMAXSOLVE(FEmparm *thee, Vio *sock) {
00367
00368     char tok[VMAX_BUFSIZE];
00369     int ti;
00370
00371     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00372     if (sscanf(tok, "%d", &ti) == 0) {
00373         Vnm_print(2, "parseFE:  read non-int (%s) for maxsolve!\n", tok);
00374         return VRC_FAILURE;
00375     }
00376     thee->maxsolve = ti;
00377     thee->setmaxsolve = 1;
00378     return VRC_SUCCESS;
00379 VERROR1:
00380     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00381     return VRC_FAILURE;
00382 }
00383 }
00384
00385 VPRIVATE Vrc_Codes FEmparm_parseMAXVERT(FEmparm *thee, Vio *sock) {
00386
00387     char tok[VMAX_BUFSIZE];
00388     int ti;
00389
00390     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00391     if (sscanf(tok, "%d", &ti) == 0) {
00392         Vnm_print(2, "parseFE:  read non-int (%s) for maxvert!\n", tok);
00393         return VRC_FAILURE;
00394     }
00395     thee->maxvert = ti;
00396     thee->setmaxvert = 1;
00397     return VRC_SUCCESS;
00398
00399 VERROR1:
00400     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00401     return VRC_FAILURE;
00402

```

```

00403 }
00404
00405 VPRIVATE Vrc_Codes FEMparm_parseUSEMESH(FEMparm *thee, Vio *sock) {
00406     char tok[VMAX_BUFSIZE];
00407     int ti;
00408
00409     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00410     if (sscanf(tok, "%d", &ti) == 0) {
00411         Vnm_print(2, "parseFE:  read non-int (%s) for usemesh!\n", tok);
00412         return VRC_FAILURE;
00413     }
00414     thee->useMesh = 1;
00415     thee->meshID = ti;
00416
00417     return VRC_SUCCESS;
00418
00419 VERROR1:
00420     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00421     return VRC_FAILURE;
00422 }
00423
00424
00425 VPUBLIC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
00426     Vio *sock) {
00427
00428     int i, ti;
00429     double tf;
00430
00431     if (thee == VNULL) {
00432         Vnm_print(2, "parseFE:  got NULL thee!\n");
00433         return VRC_FAILURE;
00434     }
00435
00436     if (sock == VNULL) {
00437         Vnm_print(2, "parseFE:  got NULL socket!\n");
00438         return VRC_FAILURE;
00439     }
00440
00441     if (Vstring_strcasecmp(tok, "domainLength") == 0) {
00442         return FEMparm_parseDOMAINLENGTH(thee, sock);
00443     } else if (Vstring_strcasecmp(tok, "etol") == 0) {
00444         return FEMparm_parseETOL(thee, sock);
00445     } else if (Vstring_strcasecmp(tok, "ekey") == 0) {
00446         return FEMparm_parseEKEY(thee, sock);
00447     } else if (Vstring_strcasecmp(tok, "akeyPRE") == 0) {
00448         return FEMparm_parseAKEYPRE(thee, sock);
00449     } else if (Vstring_strcasecmp(tok, "akeySOLVE") == 0) {
00450         return FEMparm_parseAKEYSOLVE(thee, sock);
00451     } else if (Vstring_strcasecmp(tok, "targetNum") == 0) {
00452         return FEMparm_parseTARGETNUM(thee, sock);
00453     } else if (Vstring_strcasecmp(tok, "targetRes") == 0) {
00454         return FEMparm_parseTARGETRES(thee, sock);
00455     } else if (Vstring_strcasecmp(tok, "maxsolve") == 0) {
00456         return FEMparm_parseMAXSOLVE(thee, sock);
00457     } else if (Vstring_strcasecmp(tok, "maxvert") == 0) {
00458         return FEMparm_parseMAXVERT(thee, sock);
00459     } else if (Vstring_strcasecmp(tok, "usemesh") == 0) {

```

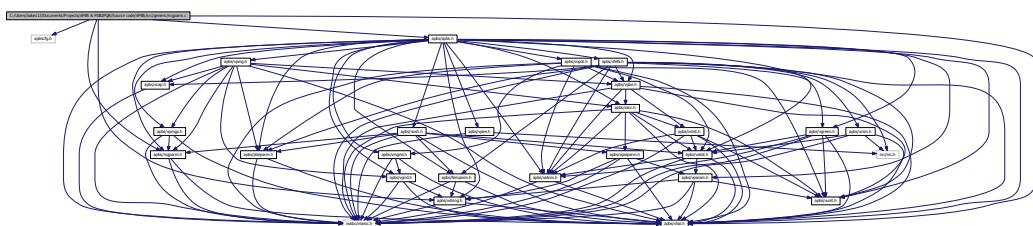
```
00460     return FEMparm_parseUSEMESH(thee, sock);
00461 }
00462
00463     return VRC_WARNING;
00464
00465 }
```

10.55 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/mgparm.c File Reference

Class MGparm methods.

```
#include "apbscfg.h"
#include "apbs/apbs.h"
#include "apbs/vhal.h"
#include "apbs/mgparm.h"
#include "apbs/vstring.h"
```

Include dependency graph for mgparm.c:



Functions

- VPUBLIC void [MGparm_setCenterX](#) (MGparm *thee, double x)
Set center x-coordinate.
- VPUBLIC void [MGparm_setCenterY](#) (MGparm *thee, double y)
Set center y-coordinate.
- VPUBLIC void [MGparm_setCenterZ](#) (MGparm *thee, double z)
Set center z-coordinate.
- VPUBLIC double [MGparm_getCenterX](#) (MGparm *thee)
Get center x-coordinate.
- VPUBLIC double [MGparm_getCenterY](#) (MGparm *thee)

- VPUBLIC double **MGparm_getCenterZ** (**MGparm** *thee)

Get center y-coordinate.
- VPUBLIC int **MGparm_getNx** (**MGparm** *thee)

Get center z-coordinate.
- VPUBLIC int **MGparm_getNy** (**MGparm** *thee)

Get number of grid points in x direction.
- VPUBLIC int **MGparm_getNz** (**MGparm** *thee)

Get number of grid points in y direction.
- VPUBLIC int **MGparm_getHx** (**MGparm** *thee)

Get number of grid points in z direction.
- VPUBLIC double **MGparm_getHy** (**MGparm** *thee)

Get grid spacing in x direction (Å)
- VPUBLIC double **MGparm_getHz** (**MGparm** *thee)

Get grid spacing in y direction (Å)
- VPUBLIC double **MGparm_getHx** (**MGparm** *thee)

Get grid spacing in z direction (Å)
- VPUBLIC **MGparm *** **MGparm_ctor** (**MGparm_CalcType** type)

Construct MGparm object.
- VPUBLIC Vrc_Codes **MGparm_ctor2** (**MGparm** *thee, **MGparm_CalcType** type)

FORTRAN stub to construct MGparm object.
- VPUBLIC void **MGparm_dtor** (**MGparm** **thee)

Object destructor.
- VPUBLIC void **MGparm_dtor2** (**MGparm** *thee)

FORTRAN stub for object destructor.
- VPUBLIC Vrc_Codes **MGparm_check** (**MGparm** *thee)

Consistency check for parameter values stored in object.
- VPUBLIC void **MGparm_copy** (**MGparm** *thee, **MGparm** *parm)

Copy MGparm object into thee.
- VPRIVATE Vrc_Codes **MGparm_parseDIME** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseCHGM** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseNLEV** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseETOL** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseGRID** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseGLEN** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseGAMMA** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseGCENT** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseCGLEN** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseFGLEN** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseCGCENT** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseFGCENT** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parsePDIME** (**MGparm** *thee, **Vio** *sock)

- VPRIVATE Vrc_Codes **MGparm_parseOFRAC** (**MGparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **MGparm_parseASYNC** (**MGparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **MGparm_parseUSEAQUA** (**MGparm** *thee, Vio *sock)
- VPUBLIC Vrc_Codes **MGparm_parseToken** (**MGparm** *thee, char tok[VMAX_BUFSIZE], Vio *sock)

Parse an MG keyword from an input file.

10.55.1 Detailed Description

Class MGparm methods.

Author

Nathan Baker

Version

Id:

[mgparm.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation
```

```

* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [mgparm.c](#).

10.56 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/mgparm.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/apbs.h"
00059 #include "apbs/vhal.h"
00060 #include "apbs/mgparm.h"
00061 #include "apbs/vstring.h"
00062
00063 VEMBED(rcsid="$Id: mgparm.c 1667 2011-12-02 23:22:02Z pcellis $")
00064
00065 #if !defined(VINLINE_MGPARM)
00066
00067 #endif /* if !defined(VINLINE_MGPARM) */
00068
00069 VPUBLIC void MGparm_setCenterX(MGparm *thee, double x) {
00070     VASSERT(thee != VNULL);
00071     thee->center[0] = x;
00072 }
00073 VPUBLIC void MGparm_setCenterY(MGparm *thee, double y) {
00074     VASSERT(thee != VNULL);
00075     thee->center[1] = y;
00076 }
00077 VPUBLIC void MGparm_setCenterZ(MGparm *thee, double z) {
00078     VASSERT(thee != VNULL);
00079     thee->center[2] = z;
00080 }
00081 VPUBLIC double MGparm_getCenterX(MGparm *thee) {
00082     VASSERT(thee != VNULL);
00083     return thee->center[0];

```

```

00084 }
00085 VPUBLIC double MGparm_getCenterY(MGparm *thee) {
00086     VASSERT(thee != VNULL);
00087     return thee->center[1];
00088 }
00089 VPUBLIC double MGparm_getCenterZ(MGparm *thee) {
00090     VASSERT(thee != VNULL);
00091     return thee->center[2];
00092 }
00093 VPUBLIC int MGparm_getNx(MGparm *thee) {
00094     VASSERT(thee != VNULL);
00095     return thee->dime[0];
00096 }
00097 VPUBLIC int MGparm_getNy(MGparm *thee) {
00098     VASSERT(thee != VNULL);
00099     return thee->dime[1];
00100 }
00101 VPUBLIC int MGparm_getNz(MGparm *thee) {
00102     VASSERT(thee != VNULL);
00103     return thee->dime[2];
00104 }
00105 VPUBLIC double MGparm_getHx(MGparm *thee) {
00106     VASSERT(thee != VNULL);
00107     return thee->grid[0];
00108 }
00109 VPUBLIC double MGparm_getHy(MGparm *thee) {
00110     VASSERT(thee != VNULL);
00111     return thee->grid[1];
00112 }
00113 VPUBLIC double MGparm_getHz(MGparm *thee) {
00114     VASSERT(thee != VNULL);
00115     return thee->grid[2];
00116 }
00117
00118 VPUBLIC MGparm* MGparm_ctor(MGparm_CalcType type) {
00119
00120     /* Set up the structure */
00121     MGparm *thee = VNULL;
00122     thee = Vmem_malloc(VNULL, 1, sizeof(MGparm));
00123     VASSERT( thee != VNULL );
00124     VASSERT( MGparm_ctor2(thee, type) == VRC_SUCCESS );
00125
00126     return thee;
00127 }
00128
00129 VPUBLIC Vrc_Codes MGparm_ctor2(MGparm *thee, MGparm_CalcType type) {
00130
00131     int i;
00132
00133     if (thee == VNULL) return VRC_FAILURE;
00134
00135     for (i=0; i<3; i++) {
00136         thee->dime[i] = -1;
00137         thee->pdime[i] = 1;
00138     }
00139
00140     thee->parsed = 0;

```

```

00141     thee->type = type;
00142
00143     /* *** GENERIC PARAMETERS *** */
00144     thee->setdime = 0;
00145     thee->setchgm = 0;
00146
00147     /* *** TYPE 0 PARAMETERS *** */
00148     thee->nlev = VMGNLEV;
00149     thee->setnlev = 1;
00150     thee->etol = 1.0e-6;
00151     thee->setetol = 0;
00152     thee->setgrid = 0;
00153     thee->setglen = 0;
00154     thee->setgcent = 0;
00155
00156     /* *** TYPE 1 & 2 PARAMETERS *** */
00157     thee->setcglen = 0;
00158     thee->setfglen = 0;
00159     thee->setcgcent = 0;
00160     thee->setfgcent = 0;
00161
00162     /* *** TYPE 2 PARAMETERS *** */
00163     thee->setpdime = 0;
00164     thee->setrank = 0;
00165     thee->setszie = 0;
00166     thee->setofrac = 0;
00167     for (i=0; i<6; i++) thee->partDisjOwnSide[i] = 0;
00168     thee->setasync = 0;
00169
00170     /* *** Default parameters for TINKER *** */
00171     thee->chgs = VCM_CHARGE;
00172
00173     thee->useAqua = 0;
00174     thee->setUseAqua = 0;
00175
00176     return VRC_SUCCESS;
00177 }
00178
00179 VPUBLIC void MGparm_dtor(MGparm **thee) {
00180     if ((*thee) != VNULL) {
00181         MGparm_dtor2(*thee);
00182         Vmem_free(VNULL, 1, sizeof(MGparm), (void **)thee);
00183         (*thee) = VNULL;
00184     }
00185 }
00186
00187 VPUBLIC void MGparm_dtor2(MGparm *thee) { ; }
00188
00189 VPUBLIC Vrc_Codes MGparm_check(MGparm *thee) {
00190
00191     Vrc_Codes rc;
00192     int i, tdime[3], ti, tnlev[3], nlev;
00193
00194     rc = VRC_SUCCESS;
00195
00196     Vnm_print(0, "MGparm_check: checking MGparm object of type %d.\n",
00197             thee->type);

```

```

00198     /* Check to see if we were even filled... */
00199     if (!thee->parsed) {
00200         Vnm_print(2, "MGparm_check:  not filled!\n");
00201         return VRC_FAILURE;
00202     }
00203
00204
00205     /* Check generic settings */
00206     if (!thee->setdime) {
00207         Vnm_print(2, "MGparm_check:  DIME not set!\n");
00208         rc = VRC_FAILURE;
00209     }
00210     if (!thee->setchgm) {
00211         Vnm_print(2, "MGparm_check:  CHGM not set!\n");
00212         return VRC_FAILURE;
00213     }
00214
00215
00216     /* Check sequential manual & dummy settings */
00217     if ((thee->type == MCT_MANUAL) || (thee->type == MCT_DUMMY)) {
00218         if ((!thee->setgrid) && (!thee->setglen)) {
00219             Vnm_print(2, "MGparm_check:  Neither GRID nor GLEN set!\n");
00220             rc = VRC_FAILURE;
00221         }
00222         if ((thee->setgrid) && (thee->setglen)) {
00223             Vnm_print(2, "MGparm_check:  Both GRID and GLEN set!\n");
00224             rc = VRC_FAILURE;
00225         }
00226         if (!thee->setgcnt) {
00227             Vnm_print(2, "MGparm_check:  GCENT not set!\n");
00228             rc = VRC_FAILURE;
00229         }
00230     }
00231
00232     /* Check sequential and parallel automatic focusing settings */
00233     if ((thee->type == MCT_AUTO) || (thee->type == MCT_PARALLEL)) {
00234         if (!thee->setcglen) {
00235             Vnm_print(2, "MGparm_check:  CGLEN not set!\n");
00236             rc = VRC_FAILURE;
00237         }
00238         if (!thee->setfglen) {
00239             Vnm_print(2, "MGparm_check:  FGLEN not set!\n");
00240             rc = VRC_FAILURE;
00241         }
00242         if (!thee->setgcnt) {
00243             Vnm_print(2, "MGparm_check:  CGCENT not set!\n");
00244             rc = VRC_FAILURE;
00245         }
00246         if (!thee->setfgcnt) {
00247             Vnm_print(2, "MGparm_check:  FGCENT not set!\n");
00248             rc = VRC_FAILURE;
00249         }
00250     }
00251
00252     /* Check parallel automatic focusing settings */
00253     if (thee->type == MCT_PARALLEL) {
00254         if (!thee->setpdime) {

```

```

00255             Vnm_print(2, "MGparm_check: PDIME not set!\n");
00256             rc = VRC_FAILURE;
00257         }
00258         if (!thee->setrank) {
00259             Vnm_print(2, "MGparm_check: PROC_RANK not set!\n");
00260             rc = VRC_FAILURE;
00261         }
00262         if (!thee->setsiz) {
00263             Vnm_print(2, "MGparm_check: PROC_SIZE not set!\n");
00264             rc = VRC_FAILURE;
00265         }
00266         if (!thee->setofrac) {
00267             Vnm_print(2, "MGparm_check: OFRAC not set!\n");
00268             rc = VRC_FAILURE;
00269         }
00270     }
00271
00272     /* Perform a sanity check on nlev and dime, resetting values as necessary */
00273     if (rc == 1) {
00274     /* Calculate the actual number of grid points and nlev to satisfy the
00275      * formula: n = c * 2^(l+1) + 1, where n is the number of grid points,
00276      * c is an integer, and l is the number of levels */
00277     if (thee->type != MCT_DUMMY) {
00278         for (i=0; i<3; i++) {
00279             /* See if the user picked a reasonable value, if not then fix it */
00280             ti = thee->dime[i] - 1;
00281             if (ti == VPOW(2, (thee->nlev+1))) {
00282                 tnlev[i] = thee->nlev;
00283                 tdime[i] = thee->dime[i];
00284             } else {
00285                 tdime[i] = thee->dime[i];
00286                 ti = tdime[i] - 1;
00287                 tnlev[i] = 0;
00288                 /* Find the maximum number of times this dimension can be
00289                  * divided by two */
00290                 while (VEVEN(ti)) {
00291                     (tnlev[i])++;
00292                     ti = (int)ceil(0.5*ti);
00293                 }
00294                 (tnlev[i])--;
00295                 /* We'd like to have at least VMGNLEV levels in the multigrid
00296                  * hierarchy. This means that the dimension needs to be
00297                  * c*2^VMGNLEV + 1, where c is an integer. */
00298                 if ((tdime[i] > 65) && (tnlev[i] < VMGNLEV)) {
00299                     Vnm_print(2, "NOsh: Bad dime[%d] = %d (%d nlev)!\n",
00300                               i, tdime[i], tnlev[i]);
00301                     ti = (int)(tdime[i]/VPOW(2., (VMGNLEV+1)));
00302                     if (ti < 1) ti = 1;
00303                     tdime[i] = ti*(int)(VPOW(2., (VMGNLEV+1))) + 1;
00304                     tnlev[i] = 4;
00305                     Vnm_print(2, "NOsh: Reset dime[%d] to %d and (nlev = %d).\n",
00306                               i, tdime[i], VMGNLEV);
00307                 }
00308             }
00309         } else { /* We are a dummy calculation, but we still need positive numbers of po
ints */

```

```

00310     for (i=0; i<3; i++) {
00311         tnlev[i] = thee->nlev;
00312         tdime[i] = thee->dime[i];
00313         if (thee->dime[i] <= 0) {
00314             Vnm_print(2, "NOsh: Resetting dime[%d] from %d to 3.\n", i, thee->dime[i]);
00315             thee->dime[i] = 3;
00316         }
00317     }
00318 }
00319
00320 /* The actual number of levels we'll be using is the smallest number of
00321      * possible levels in any dimensions */
00322 nlev = VMIN2(tnlev[0], tnlev[1]);
00323 nlev = VMIN2(nlev, tnlev[2]);
00324 /* Set the number of levels and dimensions */
00325 Vnm_print(0, "NOsh: nlev = %d, dime = (%d, %d, %d)\n", nlev, tdime[0],
00326             tdime[1], tdime[2]);
00327 thee->nlev = nlev;
00328 if (thee->nlev <= 0) {
00329     Vnm_print(2, "MGparm_check: illegal nlev (%d); check your grid dimensions!\n"
00330 , thee->nlev);
00331     rc = VRC_FAILURE;
00332 }
00333 if (thee->nlev < 2) {
00334     Vnm_print(2, "MGparm_check: you're using a very small nlev (%d) and therefore
00335 \n", thee->nlev);
00336     Vnm_print(2, "MGparm_check: will not get the optimal performance of the multi
00337 grid\n");
00338     Vnm_print(2, "MGparm_check: algorithm. Please check your grid dimensions.\n"
00339 );
00340 }
00341     for (i=0; i<3; i++) thee->dime[i] = tdime[i];
00342 }
00343 }
00344
00345 VPUBLIC void MGparm_copy(MGparm *thee, MGparm *parm) {
00346
00347     int i;
00348
00349     VASSERT(thee != VNULL);
00350     VASSERT(parm != VNULL);
00351
00352
00353     thee->type = parm->type;
00354     thee->parsed = parm->parsed;
00355
00356     /* *** GENERIC PARAMETERS *** */
00357     for (i=0; i<3; i++) thee->dime[i] = parm->dime[i];
00358     thee->setdime = parm->setdime;
00359     thee->chgm = parm->chgm;
00360     thee->setchgm = parm->setchgm;
00361     thee->chgs = parm->chgs;

```

```

00362      /* *** TYPE 0 PARMs *** */
00363      thee->nlev = parm->nlev;
00364      thee->setnlev = parm->setnlev;
00365      thee->etol = parm->etol;
00366      thee->setetol = parm->setetol;
00367      for (i=0; i<3; i++) thee->grid[i] = parm->grid[i];
00368      thee->setgrid = parm->setgrid;
00369      for (i=0; i<3; i++) thee->glen[i] = parm->glen[i];
00370      thee->setglen = parm->setglen;
00371      thee->cmeth = parm->cmeth;
00372      for (i=0; i<3; i++) thee->center[i] = parm->center[i];
00373      thee->setgcent = parm->setgcent;
00374      thee->centmol = parm->centmol;
00375
00376
00377      /* *** TYPE 1 & 2 PARMs *** */
00378      for (i=0; i<3; i++) thee->crlen[i] = parm->crlen[i];
00379      thee->setcrlen = parm->setcrlen;
00380      for (i=0; i<3; i++) thee->flen[i] = parm->flen[i];
00381      thee->setflen = parm->setflen;
00382      thee->ccmeth = parm->ccmeth;
00383      for (i=0; i<3; i++) thee->ccenter[i] = parm->ccenter[i];
00384      thee->setccent = parm->setccent;
00385      thee->ccentmol = parm->ccentmol;
00386      thee->fcmeth = parm->fcmeth;
00387      for (i=0; i<3; i++) thee->fcenter[i] = parm->fcenter[i];
00388      thee->setfgcent = parm->setfgcent;
00389      thee->fcntmol = parm->fcntmol;
00390
00391      /* *** TYPE 2 PARMs *** */
00392      for (i=0; i<3; i++)
00393          thee->partDisjCenter[i] = parm->partDisjCenter[i];
00394      for (i=0; i<3; i++)
00395          thee->partDisjLength[i] = parm->partDisjLength[i];
00396      for (i=0; i<6; i++)
00397          thee->partDisjOwnSide[i] = parm->partDisjOwnSide[i];
00398      for (i=0; i<3; i++) thee->pdime[i] = parm->pdime[i];
00399      thee->setpdime = parm->setpdime;
00400      thee->proc_rank = parm->proc_rank;
00401      thee->setrank = parm->setrank;
00402      thee->proc_size = parm->proc_size;
00403      thee->setszie = parm->setszie;
00404      thee->ofrac = parm->ofrac;
00405      thee->setofrac = parm->setofrac;
00406      thee->setasync = parm->setasync;
00407      thee->async = parm->async;
00408
00409      thee->nonlintype = parm->nonlintype;
00410      thee->setnonlintype = parm->setnonlintype;
00411
00412      thee->method = parm->method;
00413      thee->method = parm->method;
00414
00415      thee->useAqua = parm->useAqua;
00416      thee->setUseAqua = parm->setUseAqua;
00417  }
00418

```

```

00419 VPRIVATE Vrc_Codes MGparm_parseDIME(MGparm *thee, Vio *sock) {
00420
00421     char tok[VMAX_BUFSIZE];
00422     int ti;
00423
00424     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00425     if (sscanf(tok, "%d", &ti) == 0) {
00426         Vnm_print(2, "parseMG: Read non-integer (%s) while parsing DIME \
00427 keyword!\n", tok);
00428         return VRC_WARNING;
00429     } else thee->dime[0] = ti;
00430     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00431     if (sscanf(tok, "%d", &ti) == 0) {
00432         Vnm_print(2, "NOsh: Read non-integer (%s) while parsing DIME \
00433 keyword!\n", tok);
00434         return VRC_WARNING;
00435     } else thee->dime[1] = ti;
00436     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00437     if (sscanf(tok, "%d", &ti) == 0) {
00438         Vnm_print(2, "NOsh: Read non-integer (%s) while parsing DIME \
00439 keyword!\n", tok);
00440         return VRC_WARNING;
00441     } else thee->dime[2] = ti;
00442     thee->setdime = 1;
00443     return VRC_SUCCESS;
00444
00445     VERROR1:
00446     Vnm_print(2, "parseMG: ran out of tokens!\n");
00447     return VRC_WARNING;
00448 }
00449
00450 VPRIVATE Vrc_Codes MGparm_parseCHGM(MGparm *thee, Vio *sock) {
00451
00452     char tok[VMAX_BUFSIZE];
00453     Vchrg_Meth ti;
00454
00455     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00456     if (sscanf(tok, "%d", &ti) == 1) {
00457         thee->chgm = ti;
00458         thee->setchgm = 1;
00459         Vnm_print(2, "NOsh: Warning -- parsed deprecated statement \"chgm %d\".\n",
00460                  ", ti);
00461         Vnm_print(2, "NOsh: Please use \"chgm \"");
00462         switch (thee->chgm) {
00463             case VCM_TRI1:
00464                 Vnm_print(2, "spl0");
00465                 break;
00466             case VCM_BSPL2:
00467                 Vnm_print(2, "spl2");
00468                 break;
00469             case VCM_BSPL4:
00470                 Vnm_print(2, "spl4");
00471                 break;
00472             default:
00473                 Vnm_print(2, "UNKNOWN");
00474                 break;
00475         }

```

```

00475         Vnm_print(2, "\" instead!\n");
00476         return VRC_SUCCESS;
00477     } else if (Vstring_strcasecmp(tok, "spl0") == 0) {
00478         thee->chgm = VCM_TRIL;
00479         thee->setchgm = 1;
00480         return VRC_SUCCESS;
00481     } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
00482         thee->chgm = VCM_BSPL2;
00483         thee->setchgm = 1;
00484         return VRC_SUCCESS;
00485     } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
00486         thee->chgm = VCM_BSPL4;
00487         thee->setchgm = 1;
00488         return VRC_SUCCESS;
00489     } else {
00490         Vnm_print(2, "NOsh: Unrecognized parameter (%s) when parsing \
00491 chgm!\n", tok);
00492         return VRC_WARNING;
00493     }
00494     return VRC_WARNING;
00495
00496     VERROR1:
00497     Vnm_print(2, "parseMG: ran out of tokens!\n");
00498     return VRC_WARNING;
00499 }
00500
00501 VPRIVATE Vrc_Codes MGparm_parseNLEV(MGparm *thee, Vio *sock) {
00502
00503     char tok[VMAX_BUFSIZE];
00504     int ti;
00505
00506     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00507     if (sscanf(tok, "%d", &ti) == 0) {
00508         Vnm_print(2, "NOsh: Read non-integer (%s) while parsing NLEV \
00509 keyword!\n", tok);
00510         return VRC_WARNING;
00511     } else thee->nlev = ti;
00512     thee->setnlev = 1;
00513     return VRC_SUCCESS;
00514
00515     VERROR1:
00516     Vnm_print(2, "parseMG: ran out of tokens!\n");
00517     return VRC_WARNING;
00518 }
00519
00520 VPRIVATE Vrc_Codes MGparm_parseETOL(MGparm *thee, Vio *sock) {
00521
00522     char tok[VMAX_BUFSIZE];
00523     double tf;
00524
00525     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00526     if (sscanf(tok, "%lf", &tf) == 0) {
00527         Vnm_print(2, "NOsh: Read non-float (%s) while parsing etol \
00528 keyword!\n", tok);
00529         return VRC_WARNING;
00530     } else if (tf <= 0.0) {
00531         Vnm_print(2, "parseMG: etol must be greater than 0!\n");

```

```

00532     return VRC_WARNING;
00533 } else thee->etol = tf;
00534 thee->setetol = 1;
00535 return VRC_SUCCESS;
00536
00537 VERROR1:
00538     Vnm_print(2, "parseMG: ran out of tokens!\n");
00539     return VRC_WARNING;
00540 }
00541
00542
00543 VPRIVATE Vrc_Codes MGparm_parseGRID(MGparm *thee, Vio *sock) {
00544
00545     char tok[VMAX_BUFSIZE];
00546     double tf;
00547
00548     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00549     if (sscanf(tok, "%lf", &tf) == 0) {
00550         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GRID \
00551 keyword!\n", tok);
00552         return VRC_WARNING;
00553     } else thee->grid[0] = tf;
00554     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00555     if (sscanf(tok, "%lf", &tf) == 0) {
00556         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GRID \
00557 keyword!\n", tok);
00558         return VRC_WARNING;
00559     } else thee->grid[1] = tf;
00560     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00561     if (sscanf(tok, "%lf", &tf) == 0) {
00562         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GRID \
00563 keyword!\n", tok);
00564         return VRC_WARNING;
00565     } else thee->grid[2] = tf;
00566     thee->setgrid = 1;
00567     return VRC_SUCCESS;
00568
00569 VERROR1:
00570     Vnm_print(2, "parseMG: ran out of tokens!\n");
00571     return VRC_WARNING;
00572 }
00573
00574 VPRIVATE Vrc_Codes MGparm_parseGLEN(MGparm *thee, Vio *sock) {
00575
00576     char tok[VMAX_BUFSIZE];
00577     double tf;
00578
00579     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00580     if (sscanf(tok, "%lf", &tf) == 0) {
00581         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GLEN \
00582 keyword!\n", tok);
00583         return VRC_WARNING;
00584     } else thee->glen[0] = tf;
00585     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00586     if (sscanf(tok, "%lf", &tf) == 0) {
00587         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GLEN \
00588 keyword!\n", tok);

```

```

00589         return VRC_WARNING;
00590     } else thee->glen[1] = tf;
00591     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00592     if (sscanf(tok, "%lf", &tf) == 0) {
00593         Vnm_print(2, "NOsh: Read non-float (%s) while parsing GLEN \
00594 keyword!\n", tok);
00595         return VRC_WARNING;
00596     } else thee->glen[2] = tf;
00597     thee->setglen = 1;
00598     return VRC_SUCCESS;
00599
00600     VERROR1:
00601         Vnm_print(2, "parseMG: ran out of tokens!\n");
00602         return VRC_WARNING;
00603 }
00604
00605 VPRIVATE Vrc_Codes MGparm_parseGAMMA(MGparm *thee, Vio *sock) {
00606
00607     char tok[VMAX_BUFSIZE];
00608
00609     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00610     Vnm_print(2, "parseMG: GAMMA keyword deprecated!\n");
00611     Vnm_print(2, "parseMG: If you are using PyMOL or VMD and still seeing this mess \
00612 age,\n");
00613     Vnm_print(2, "parseMG: please contact the developers of those programs regardin \
00614 g this message.\n");
00615     return VRC_SUCCESS;
00616
00617     VERROR1:
00618         Vnm_print(2, "parseMG: ran out of tokens!\n");
00619         return VRC_WARNING;
00620 }
00621
00622 VPRIVATE Vrc_Codes MGparm_parseGCENT(MGparm *thee, Vio *sock) {
00623
00624     char tok[VMAX_BUFSIZE];
00625     double tf;
00626     int ti;
00627
00628     /* If the next token isn't a float, it probably means we want to \
00629      * center on a molecule */
00630     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00631     if (sscanf(tok, "%lf", &tf) == 0) {
00632         if (Vstring_strcasecmp(tok, "mol") == 0) {
00633             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00634             if (sscanf(tok, "%d", &ti) == 0) {
00635                 Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00636 GCENT MOL keyword!\n", tok);
00637                 return VRC_WARNING;
00638             } else {
00639                 thee->cmeth = MCM_MOLECULE;
00640                 /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into \
00641                  array index */
00642                 thee->centmol = ti - 1;
00643             }
00644         } else {
00645             Vnm_print(2, "NOsh: Unexpected keyword (%s) while parsing \

```

```

00644 GCENT!\n", tok);
00645         return VRC_WARNING;
00646     }
00647 } else {
00648     thee->center[0] = tf;
00649     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00650     if (sscanf(tok, "%lf", &tf) == 0) {
00651         Vnm_print(2, "NOsh: Read non-float (%s) while parsing \
00652 GCENT keyword!\n", tok);
00653         return VRC_WARNING;
00654     }
00655     thee->center[1] = tf;
00656     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00657     if (sscanf(tok, "%lf", &tf) == 0) {
00658         Vnm_print(2, "NOsh: Read non-float (%s) while parsing \
00659 GCENT keyword!\n", tok);
00660         return VRC_WARNING;
00661     }
00662     thee->center[2] = tf;
00663 }
00664 thee->setgcent = 1;
00665 return VRC_SUCCESS;
00666
00667 VERROR1:
00668     Vnm_print(2, "parseMG: ran out of tokens!\n");
00669     return VRC_WARNING;
0070 }

0072 VPRIVATE Vrc_Codes MGparm_parseCGLEN(MGparm *thee, Vio *sock) {
0073
0074     char tok[VMAX_BUFSIZE];
0075     double tf;
0076
0077     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
0078     if (sscanf(tok, "%lf", &tf) == 0) {
0079         Vnm_print(2, "NOsh: Read non-float (%s) while parsing CGLEN \
0080 keyword!\n", tok);
0081         return VRC_WARNING;
0082     } else thee->crlen[0] = tf;
0083     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
0084     if (sscanf(tok, "%lf", &tf) == 0) {
0085         Vnm_print(2, "NOsh: Read non-float (%s) while parsing CGLEN \
0086 keyword!\n", tok);
0087         return VRC_WARNING;
0088     } else thee->crlen[1] = tf;
0089     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
0090     if (sscanf(tok, "%lf", &tf) == 0) {
0091         Vnm_print(2, "NOsh: Read non-float (%s) while parsing CGLEN \
0092 keyword!\n", tok);
0093         return VRC_WARNING;
0094     } else thee->crlen[2] = tf;
0095     thee->setcrlen = 1;
0096     return VRC_SUCCESS;
0097
0098 VERROR1:
0099     Vnm_print(2, "parseMG: ran out of tokens!\n");
0100     return VRC_WARNING;

```

```

00701 }
00702
00703 VPRIVATE Vrc_Codes MGparm_parseFGLEN (MGparm *thee, Vio *sock) {
00704
00705     char tok[VMAX_BUFSIZE];
00706     double tf;
00707
00708     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00709     if (sscanf(tok, "%lf", &tf) == 0) {
00710         Vnm_print(2, "NOSH: Read non-float (%s) while parsing FGLEN \
00711 keyword!\n", tok);
00712         return VRC_WARNING;
00713     } else thee->fglen[0] = tf;
00714     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00715     if (sscanf(tok, "%lf", &tf) == 0) {
00716         Vnm_print(2, "NOSH: Read non-float (%s) while parsing FGLEN \
00717 keyword!\n", tok);
00718         return VRC_WARNING;
00719     } else thee->fglen[1] = tf;
00720     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00721     if (sscanf(tok, "%lf", &tf) == 0) {
00722         Vnm_print(2, "NOSH: Read non-float (%s) while parsing FGLEN \
00723 keyword!\n", tok);
00724         return VRC_WARNING;
00725     } else thee->fglen[2] = tf;
00726     thee->setfglen = 1;
00727     return VRC_SUCCESS;
00728
00729     VERROR1:
00730         Vnm_print(2, "parseMG: ran out of tokens!\n");
00731         return VRC_WARNING;
00732 }
00733
00734 VPRIVATE Vrc_Codes MGparm_parseCGCENT (MGparm *thee, Vio *sock) {
00735
00736     char tok[VMAX_BUFSIZE];
00737     double tf;
00738     int ti;
00739
00740     /* If the next token isn't a float, it probably means we want to
00741      * center on a molecule */
00742     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00743     if (sscanf(tok, "%lf", &tf) == 0) {
00744         if (Vstring_strcasecmp(tok, "mol") == 0) {
00745             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00746             if (sscanf(tok, "%d", &ti) == 0) {
00747                 Vnm_print(2, "NOSH: Read non-int (%s) while parsing \
00748 CGCENT MOL keyword!\n", tok);
00749                 return VRC_WARNING;
00750             } else {
00751                 thee->ccmeth = MCM_MOLECULE;
00752                 /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00753                  array index */
00754                 thee->ccentmol = ti - 1;
00755             }
00756         } else {
00757             Vnm_print(2, "NOSH: Unexpected keyword (%s) while parsing \

```

```

00758 CGCENT!\n", tok);
00759         return VRC_WARNING;
00760     }
00761 } else {
00762     thee->ccenter[0] = tf;
00763     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00764     if (sscanf(tok, "%lf", &tf) == 0) {
00765         Vnm_print(2, "NOsh: Read non-float (%s) while parsing \
00766 CGCENT keyword!\n", tok);
00767         return VRC_WARNING;
00768     }
00769     thee->ccenter[1] = tf;
00770     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00771     if (sscanf(tok, "%lf", &tf) == 0) {
00772         Vnm_print(2, "NOsh: Read non-float (%s) while parsing \
00773 CGCENT keyword!\n", tok);
00774         return VRC_WARNING;
00775     }
00776     thee->ccenter[2] = tf;
00777 }
00778 thee->setcgcent = 1;
00779 return VRC_SUCCESS;
00780
00781 VERROR1:
00782     Vnm_print(2, "parseMG: ran out of tokens!\n");
00783     return VRC_WARNING;
00784 }
00785
00786 VPRIVATE Vrc_Codes MGparm_parseFGCENT(MGparm *thee, Vio *sock) {
00787
00788     char tok[VMAX_BUFSIZE];
00789     double tf;
00790     int ti;
00791
00792     /* If the next token isn't a float, it probably means we want to
00793      * center on a molecule */
00794     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00795     if (sscanf(tok, "%lf", &tf) == 0) {
00796         if (Vstring_strcasecmp(tok, "mol") == 0) {
00797             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00798             if (sscanf(tok, "%d", &ti) == 0) {
00799                 Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00800 FGCENT MOL keyword!\n", tok);
00801                 return VRC_WARNING;
00802             } else {
00803                 thee->fcmeth = MCM_MOLECULE;
00804             /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00805              array index */
00806             thee->fcentmol = ti - 1;
00807             }
00808         } else {
00809             Vnm_print(2, "NOsh: Unexpected keyword (%s) while parsing \
00810 FGCENT!\n", tok);
00811             return VRC_WARNING;
00812         }
00813     } else {
00814         thee->fccenter[0] = tf;

```

```

00815      VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00816      if (sscanf(tok, "%lf", &tf) == 0) {
00817          Vnm_print(2, "NOsh: Read non-float (%s) while parsing \
00818 FGCENT keyword!\n", tok);
00819          return VRC_WARNING;
00820      }
00821      thee->fcenter[1] = tf;
00822      VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00823      if (sscanf(tok, "%lf", &tf) == 0) {
00824          Vnm_print(2, "NOsh: Read non-float (%s) while parsing \
00825 FGCENT keyword!\n", tok);
00826          return VRC_WARNING;
00827      }
00828      thee->fcenter[2] = tf;
00829  }
00830  thee->setfgcent = 1;
00831  return VRC_SUCCESS;
00832
00833  VERROR1:
00834      Vnm_print(2, "parseMG: ran out of tokens!\n");
00835      return VRC_WARNING;
00836 }
00837
00838 VPRIVATE Vrc_Codes MGparm_parsePDIME (MGparm *thee, Vio *sock) {
00839
00840     char tok[VMAX_BUFSIZE];
00841     int ti;
00842
00843     /* Read the number of grid points */
00844     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00845     if (sscanf(tok, "%d", &ti) == 0) {
00846         Vnm_print(2, "NOsh: Read non-integer (%s) while parsing PDIME \
00847 keyword!\n", tok);
00848         return VRC_WARNING;
00849     } else {
00850         thee->pdime[0] = ti;
00851     }
00852     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00853     if (sscanf(tok, "%d", &ti) == 0) {
00854         Vnm_print(2, "NOsh: Read non-integer (%s) while parsing PDIME \
00855 keyword!\n", tok);
00856         return VRC_WARNING;
00857     } else {
00858         thee->pdime[1] = ti;
00859     }
00860     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00861     if (sscanf(tok, "%d", &ti) == 0) {
00862         Vnm_print(2, "NOsh: Read non-integer (%s) while parsing PDIME \
00863 keyword!\n", tok);
00864         return VRC_WARNING;
00865     } else {
00866         thee->pdime[2] = ti;
00867     }
00868     thee->setpdime = 1;
00869     return VRC_SUCCESS;
00870
00871  VERROR1:

```

```

00872         Vnm_print(2, "parseMG: ran out of tokens!\n");
00873         return VRC_WARNING;
00874     }
00875
00876 VPRIVATE Vrc_Codes MGparm_parseOFRAC(MGparm *thee, Vio *sock) {
00877
00878     char tok[VMAX_BUFSIZE];
00879     double tf;
00880
00881     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00882     if (sscanf(tok, "%lf", &tf) == 0) {
00883         Vnm_print(2, "NOsh: Read non-int (%s) while parsing OFRAC \
00884 keyword!\n", tok);
00885         return VRC_WARNING;
00886     }
00887     thee->ofrac = tf;
00888     thee->setofrac = 1;
00889     return VRC_SUCCESS;
00890
00891     VERROR1:
00892         Vnm_print(2, "parseMG: ran out of tokens!\n");
00893         return VRC_WARNING;
00894     }
00895
00896 VPRIVATE Vrc_Codes MGparm_parseASYNC(MGparm *thee, Vio *sock) {
00897
00898     char tok[VMAX_BUFSIZE];
00899     int ti;
00900
00901     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00902     if (sscanf(tok, "%i", &ti) == 0) {
00903         Vnm_print(2, "NOsh: Read non-integer (%s) while parsing ASYNC \
00904 keyword!\n", tok);
00905         return VRC_WARNING;
00906     }
00907     thee->async = ti;
00908     thee->setasync = 1;
00909     return VRC_SUCCESS;
00910
00911     VERROR1:
00912         Vnm_print(2, "parseMG: ran out of tokens!\n");
00913         return VRC_WARNING;
00914     }
00915
00916 VPRIVATE Vrc_Codes MGparm_parseUSEAQUA(MGparm *thee, Vio *sock) {
00917     Vnm_print(0, "NOsh: parsed useaqua\n");
00918     thee->useAqua = 1;
00919     thee->setUseAqua = 1;
00920     return VRC_SUCCESS;
00921 }
00922
00923 VPUBLIC Vrc_Codes MGparm_parseToken(MGparm *thee, char tok[VMAX_BUFSIZE],
00924     Vio *sock) {
00925
00926     if (thee == VNULL) {
00927         Vnm_print(2, "parseMG: got NULL thee!\n");
00928         return VRC_WARNING;

```

```

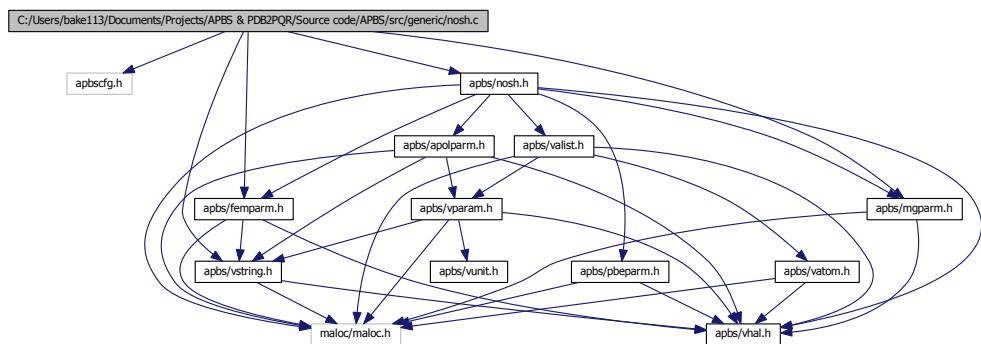
00929     }
00930     if (sock == VNULL) {
00931         Vnm_print(2, "parseMG: got NULL socket!\n");
00932         return VRC_WARNING;
00933     }
00934
00935     Vnm_print(0, "MGparm_parseToken: trying %s...\n", tok);
00936
00937
00938     if (Vstring_strcasecmp(tok, "dime") == 0) {
00939         return MGparm_parseDIME(thee, sock);
00940     } else if (Vstring_strcasecmp(tok, "chgm") == 0) {
00941         return MGparm_parseCHGM(thee, sock);
00942     } else if (Vstring_strcasecmp(tok, "nlev") == 0) {
00943         Vnm_print(2, "Warning: The 'nlev' keyword is now deprecated!\n");
00944         return MGparm_parseNLEV(thee, sock);
00945     } else if (Vstring_strcasecmp(tok, "etol") == 0) {
00946         return MGparm_parseETOL(thee, sock);
00947     } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00948         return MGparm_parseGRID(thee, sock);
00949     } else if (Vstring_strcasecmp(tok, "glen") == 0) {
00950         return MGparm_parseGLEN(thee, sock);
00951     } else if (Vstring_strcasecmp(tok, "gcent") == 0) {
00952         return MGparm_parseGCENT(thee, sock);
00953     } else if (Vstring_strcasecmp(tok, "cglen") == 0) {
00954         return MGparm_parseCGLEN(thee, sock);
00955     } else if (Vstring_strcasecmp(tok, "fglen") == 0) {
00956         return MGparm_parseFGLEN(thee, sock);
00957     } else if (Vstring_strcasecmp(tok, "cgcent") == 0) {
00958         return MGparm_parseCGCENT(thee, sock);
00959     } else if (Vstring_strcasecmp(tok, "fgcent") == 0) {
00960         return MGparm_parseFGCENT(thee, sock);
00961     } else if (Vstring_strcasecmp(tok, "pdime") == 0) {
00962         return MGparm_parsePDIME(thee, sock);
00963     } else if (Vstring_strcasecmp(tok, "ofrac") == 0) {
00964         return MGparm_parseOFRAC(thee, sock);
00965     } else if (Vstring_strcasecmp(tok, "async") == 0) {
00966         return MGparm_parseASYNC(thee, sock);
00967     } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
00968         return MGparm_parseGAMMA(thee, sock);
00969     } else if (Vstring_strcasecmp(tok, "useaqua") == 0) {
00970         return MGparm_parseUSEAQUA(thee, sock);
00971     } else {
00972         Vnm_print(2, "parseMG: Unrecognized keyword (%s)!\n", tok);
00973         return VRC_WARNING;
00974     }
00975
00976     return VRC_FAILURE;
00977
00978 }
```

10.57 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/nosh.c File Reference

Class NOsh methods.

```
#include "apbscfg.h"
#include "apbs/nosh.h"
#include "apbs/vstring.h"
#include "apbs/mgparm.h"
#include "apbs/femparm.h"
```

Include dependency graph for nosh.c:



Functions

- VPRIVATE int **NOsh_parseREAD** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parsePRINT** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parseELEC** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parseAPOLAR** (NOsh *thee, Vio *sock)
- VEXTERNC int **NOsh_parseFEM** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VEXTERNC int **NOsh_parseMG** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VEXTERNC int **NOsh_parseAPOL** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcMG** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcMGAUTO** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcMGMANUAL** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcMGPARA** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcFEM** (NOsh *thee, NOsh_calc *elec)

- VPRIVATE int **NOsh_setupCalcFEMANUAL** (**NOsh** *thee, **NOsh_calc** *elec)
- VPRIVATE int **NOsh_setupCalcAPOL** (**NOsh** *thee, **NOsh_calc** *elec)
- VPUBLIC char * **NOsh_getMolpath** (**NOsh** *thee, int imol)

Returns path to specified molecule.
- VPUBLIC char * **NOsh_getDielXpath** (**NOsh** *thee, int imol)

Returns path to specified x-shifted dielectric map.
- VPUBLIC char * **NOsh_getDielYpath** (**NOsh** *thee, int imol)

Returns path to specified y-shifted dielectric map.
- VPUBLIC char * **NOsh_getDielZpath** (**NOsh** *thee, int imol)

Returns path to specified z-shifted dielectric map.
- VPUBLIC char * **NOsh_getKappapath** (**NOsh** *thee, int imol)

Returns path to specified kappa map.
- VPUBLIC char * **NOsh_getPotpath** (**NOsh** *thee, int imol)

Returns path to specified potential map.
- VPUBLIC char * **NOsh_getChargepath** (**NOsh** *thee, int imol)

Returns path to specified charge distribution map.
- VPUBLIC **NOsh_calc** * **NOsh_getCalc** (**NOsh** *thee, int icalc)

Returns specified calculation object.
- VPUBLIC int **NOsh_getDielfmt** (**NOsh** *thee, int i)

Returns format of specified dielectric map.
- VPUBLIC int **NOsh_getKappafmt** (**NOsh** *thee, int i)

Returns format of specified kappa map.
- VPUBLIC int **NOsh_getPotfmt** (**NOsh** *thee, int i)

Returns format of specified potential map.
- VPUBLIC int **NOsh_getChargefmt** (**NOsh** *thee, int i)

Returns format of specified charge map.
- VPUBLIC **NOsh_PrintType** **NOsh_printWhat** (**NOsh** *thee, int iprint)

Return an integer ID of the observable to print .
- VPUBLIC int **NOsh_printNarg** (**NOsh** *thee, int iprint)

Return number of arguments to PRINT statement .
- VPUBLIC int **NOsh_elec2calc** (**NOsh** *thee, int icalc)

Return the name of an elec statement.
- VPUBLIC int **NOsh_apol2calc** (**NOsh** *thee, int icalc)

Return the name of an apol statement.
- VPUBLIC char * **NOsh_elecname** (**NOsh** *thee, int ielec)

Return an integer mapping of an ELEC statement to a calculation ID .
- VPUBLIC int **NOsh_printOp** (**NOsh** *thee, int iprint, int iarg)

Return integer ID for specified operation .
- VPUBLIC int **NOsh_printCalc** (**NOsh** *thee, int iprint, int iarg)

Return calculation ID for specified PRINT statement .

- VPUBLIC NOsh * NOsh_ctor (int rank, int size)
Construct NOsh.
- VPUBLIC int NOsh_ctor2 (NOsh *thee, int rank, int size)
FORTRAN stub to construct NOsh.
- VPUBLIC void NOsh_dtor (NOsh **thee)
Object destructor.
- VPUBLIC void NOsh_dtor2 (NOsh *thee)
FORTRAN stub for object destructor.
- VPUBLIC NOsh_calc * NOsh_calc_ctor (NOsh_CalcType calctype)
Construct NOsh_calc.
- VPUBLIC void NOsh_calc_dtor (NOsh_calc **thee)
Object destructor.
- VPUBLIC int NOsh_calc_copy (NOsh_calc *thee, NOsh_calc *source)
Copy NOsh_calc object into thee.
- VPUBLIC int NOsh_parseInputFile (NOsh *thee, char *filename)
Parse an input file only from a file.
- VPUBLIC int NOsh_parseInput (NOsh *thee, Vio *sock)
Parse an input file from a socket.
- VPRIVATE int NOsh_parseREAD_MOL (NOsh *thee, Vio *sock)
- VPRIVATE int NOsh_parseREAD_PARM (NOsh *thee, Vio *sock)
- VPRIVATE int NOsh_parseREAD_DIEL (NOsh *thee, Vio *sock)
- VPRIVATE int NOsh_parseREAD_KAPPA (NOsh *thee, Vio *sock)
- VPRIVATE int NOsh_parseREAD_POTENTIAL (NOsh *thee, Vio *sock)
- VPRIVATE int NOsh_parseREAD_CHARGE (NOsh *thee, Vio *sock)
- VPRIVATE int NOsh_parseREAD_MESH (NOsh *thee, Vio *sock)
- VPUBLIC int NOsh_setupElecCalc (NOsh *thee, Valist *alist[NOSH_MAXMOL])

Setup the series of electrostatics calculations.
- VPUBLIC int NOsh_setupApolCalc (NOsh *thee, Valist *alist[NOSH_MAXMOL])

Setup the series of non-polar calculations.

10.57.1 Detailed Description

Class NOsh methods.

Author

Nathan Baker

Version

Id:

[nosh.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [nosh.c](#).

10.58 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/nosh.c

```
00001
00058 #include "apbscfg.h"
00059 #include "apbs/nosh.h"
00060 #include "apbs/vstring.h"
00061 #include "apbs/mgparm.h"
00062 #include "apbs/femparm.h"
00063
00064 VEMBED(rcsid="$Id: nosh.c 1667 2011-12-02 23:22:02Z pcellis $")
00065
00066
00067 VPRIvATE int NOsh_parseREAD(
00068     NOsh *thee,
00069     Vio *sock);
00070
00071 VPRIvATE int NOsh_parsePRINT(
00072     NOsh *thee,
00073     Vio *sock);
00074
00075 VPRIvATE int NOsh_parseELEC(
00076     NOsh *thee,
00077     Vio *sock
00078 );
00079
00080 VPRIvATE int NOsh_parseAPOLAR(
00081     NOsh *thee,
00082     Vio *sock
00083 );
00084
00085 VEXTERNC int NOsh_parseFEM(
00086     NOsh *thee,
00087     Vio *sock,
00088     NOsh_calc *elec
00089 );
00090
00091 VEXTERNC int NOsh_parseMG(
00092     NOsh *thee,
00093     Vio *sock,
00094     NOsh_calc *elec
00095 );
00096
00097 VEXTERNC int NOsh_parseAPOL(
00098     NOsh *thee,
00099     Vio *sock,
00100     NOsh_calc *elec
00101 );
00102
00103 VPRIvATE int NOsh_setupCalcMG(
00104     NOsh *thee,
00105     NOsh_calc *elec
00106 );
00107
00108 VPRIvATE int NOsh_setupCalcMGAUTO(
00109     NOsh *thee,
```

```

00110         NOsh_calc *elec
00111     );
00112
00113 VPRIvATE int NOsh_setupCalcMGMANUAL(
00114     NOsh *thee,
00115     NOsh_calc *elec
00116 );
00117
00118 VPRIvATE int NOsh_setupCalcMGPARA(
00119     NOsh *thee,
00120     NOsh_calc *elec
00121 );
00122
00123 VPRIvATE int NOsh_setupCalcFEM(
00124     NOsh *thee,
00125     NOsh_calc *elec
00126 );
00127
00128 VPRIvATE int NOsh_setupCalcFEMANUAL(
00129     NOsh *thee,
00130     NOsh_calc *elec
00131 );
00132
00133 VPRIvATE int NOsh_setupCalcAPOL(
00134     NOsh *thee,
00135     NOsh_calc *elec
00136 );
00137
00138 #if !defined(VINLINE_NOSH)
00139
00140 VPUBLIC char* NOsh_getMolpath(NOsh *thee, int imol) {
00141     VASSERT(thee != VNULL);
00142     VASSERT(imol < thee->nmol);
00143     return thee->molpath[imol];
00144 }
00145 VPUBLIC char* NOsh_getDielXpath(NOsh *thee, int imol) {
00146     VASSERT(thee != VNULL);
00147     VASSERT(imol < thee->nmol);
00148     return thee->dielXpath[imol];
00149 }
00150 VPUBLIC char* NOsh_getDielYpath(NOsh *thee, int imol) {
00151     VASSERT(thee != VNULL);
00152     VASSERT(imol < thee->nmol);
00153     return thee->dielYpath[imol];
00154 }
00155 VPUBLIC char* NOsh_getDielZpath(NOsh *thee, int imol) {
00156     VASSERT(thee != VNULL);
00157     VASSERT(imol < thee->nmol);
00158     return thee->dielZpath[imol];
00159 }
00160 VPUBLIC char* NOsh_getKappapath(NOsh *thee, int imol) {
00161     VASSERT(thee != VNULL);
00162     VASSERT(imol < thee->nmol);
00163     return thee->kappapath[imol];
00164 }
00165 VPUBLIC char* NOsh_getPotpath(NOsh *thee, int imol) {
00166     VASSERT(thee != VNULL);

```

```

00167     VASSERT(imol < thee->nmol);
00168     return thee->potpath[imol];
00169 }
00170 VPUBLIC char* NOsh_getChargepath(NOsh *thee, int imol) {
00171     VASSERT(thee != VNULL);
00172     VASSERT(imol < thee->nmol);
00173     return thee->chargepath[imol];
00174 }
00175 VPUBLIC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc) {
00176     VASSERT(thee != VNULL);
00177     VASSERT(icalc < thee->nalc);
00178     return thee->calc[icalc];
00179 }
00180 VPUBLIC int NOsh_getDiefmt(NOsh *thee, int i) {
00181     VASSERT(thee != VNULL);
00182     VASSERT(i < thee->ndiel);
00183     return (thee->diefmt[i]);
00184 }
00185 VPUBLIC int NOsh_getKappafmt(NOsh *thee, int i) {
00186     VASSERT(thee != VNULL);
00187     VASSERT(i < thee->nkappa);
00188     return (thee->kappafmt[i]);
00189 }
00190 VPUBLIC int NOsh_getPotfmt(NOsh *thee, int i) {
00191     VASSERT(thee != VNULL);
00192     VASSERT(i < thee->npot);
00193     return (thee->potfmt[i]);
00194 }
00195 VPUBLIC int NOsh_getChargefmt(NOsh *thee, int i) {
00196     VASSERT(thee != VNULL);
00197     VASSERT(i < thee->ncharge);
00198     return (thee->chargefmt[i]);
00199 }
00200
00201
00202 #endif /* if !defined(VINLINE_NOSH) */
00203
00204 VPUBLIC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint) {
00205     VASSERT(thee != VNULL);
00206     VASSERT(iprint < thee->nprint);
00207     return thee->printwhat[iprint];
00208 }
00209
00210 VPUBLIC int NOsh_printNarg(NOsh *thee, int iprint) {
00211     VASSERT(thee != VNULL);
00212     VASSERT(iprint < thee->nprint);
00213     return thee->printnarg[iprint];
00214 }
00215
00216 VPUBLIC int NOsh_elec2calc(NOsh *thee, int icalc) {
00217     VASSERT(thee != VNULL);
00218     VASSERT(icalc < thee->nalc);
00219     return thee->elec2calc[icalc];
00220 }
00221
00222 VPUBLIC int NOsh_apol2calc(NOsh *thee, int icalc) {
00223     VASSERT(thee != VNULL);

```

```

00224 VASSERT(icalc < thee->nalc);
00225 return thee->apol2calc[icalc];
00226 }
00227
00228 VPUBLIC char* NOsh_elecname(NOsh *thee, int ielec) {
00229 VASSERT(thee != VNULL);
00230 VASSERT(ielec < thee->nelec + 1);
00231 return thee->elecname[ielec];
00232 }
00233
00234 VPUBLIC int NOsh_printOp(NOsh *thee, int iprint, int iarg) {
00235 VASSERT(thee != VNULL);
00236 VASSERT(iprint < thee->nprint);
00237 VASSERT(iarg < thee->printnarg[iprint]);
00238 return thee->printop[iprint][iarg];
00239 }
00240
00241 VPUBLIC int NOsh_printCalc(NOsh *thee, int iprint, int iarg) {
00242 VASSERT(thee != VNULL);
00243 VASSERT(iprint < thee->nprint);
00244 VASSERT(iarg < thee->printnarg[iprint]);
00245 return thee->printcalc[iprint][iarg];
00246 }
00247
00248 VPUBLIC NOsh* NOsh_ctor(int rank, int size) {
00249 /* Set up the structure */
00250 NOsh *thee = VNULL;
00251 thee = Vmem_malloc(VNULL, 1, sizeof(NOsh) );
00252 VASSERT( thee != VNULL);
00253 VASSERT( NOsh_ctor2(thee, rank, size) );
00254
00255 return thee;
00256 }
00257
00258
00259 VPUBLIC int NOsh_ctor2(NOsh *thee, int rank, int size) {
00260
00261 int i;
00262
00263 if (thee == VNULL) return 0;
00264
00265 thee->proc_rank = rank;
00266 thee->proc_size = size;
00267
00268 thee->ispara = 0;
00269     thee->parsed = 0;
00270
00271     thee->nmol = 0;
00272     thee->gotparm = 0;
00273     thee->ncharge = 0;
00274     thee->ndiel = 0;
00275     thee->nkappa = 0;
00276     thee->npot = 0;
00277     thee->nprint = 0;
00278
00279     for (i=0; i<NOSH_MAXCALC; i++) {
00280     thee->calc[i] = VNULL;

```

```

00281     thee->elec[i] = VNULL;
00282     thee->apol[i] = VNULL;
00283 }
00284 for (i=0; i<NOSH_MAXMOL; i++) {
00285     thee->alist[i] = VNULL;
00286 }
00287 thee->nalc = 0;
00288 thee->nelec = 0;
00289 thee->nopol = 0;
00290
00291     return 1;
00292 }
00293
00294 VPUBLIC void NOsh_dtor(NOsh **thee) {
00295     if ((*thee) != VNULL) {
00296         NOsh_dtor2(*thee);
00297         Vmem_free(VNULL, 1, sizeof(NOsh), (void **)thee);
00298         (*thee) = VNULL;
00299     }
00300 }
00301
00302 VPUBLIC void NOsh_dtor2(NOsh *thee) {
00303
00304     int i;
00305
00306     if (thee != VNULL) {
00307         for (i=0; i<(thee->nalc); i++) NOsh_calc_dtor(&(thee->calc[i]));
00308         for (i=0; i<(thee->nelec); i++) NOsh_calc_dtor(&(thee->elec[i]));
00309         for (i=0; i<(thee->nopol); i++) NOsh_calc_dtor(&(thee->apol[i]));
00310     }
00311
00312 }
00313
00314 VPUBLIC NOsh_calc* NOsh_calc_ctor(
00315     NOsh_CalcType calctype
00316 ) {
00317     NOsh_calc *thee;
00318     thee = (NOsh_calc *)Vmem_malloc(VNULL, 1, sizeof(NOsh_calc));
00319     thee->calctype = calctype;
00320     switch (calctype) {
00321     case NCT_MG:
00322         thee->mparm = MGparm_ctor(MCT_NONE);
00323         thee->fparm = VNULL;
00324         thee->apolparm = VNULL;
00325         break;
00326     case NCT_FEM:
00327         thee->mparm = VNULL;
00328         thee->fparm = FEMparm_ctor(FCT_NONE);
00329         thee->apolparm = VNULL;
00330         break;
00331     case NCT_APOL:
00332         thee->mparm = VNULL;
00333         thee->fparm = VNULL;
00334         thee->apolparm = APOLparm_ctor();
00335         break;
00336     default:
00337         Vnm_print(2, "NOsh_calc_ctor: unknown calculation type (%d)!\n",

```

```

00338         calctype);
00339     VASSERT(0);
00340 }
00341 thee->pbeparm = PBparm_ctor();
00342
00343 return thee;
00344 }
00345
00346 VPUBLIC void NOsh_calc_dtor(
00347     NOsh_calc **thee
00348 ) {
00349
00350 NOsh_calc *calc = VNULL;
00351 calc = *thee;
00352 if (calc == VNULL) return;
00353
00354 switch (calc->calctype) {
00355     case NCT_MG:
00356         MGparm_dtor(&(calc->mgparm));
00357         break;
00358     case NCT_FEM:
00359         FEMparm_dtor(&(calc->femparm));
00360         break;
00361     case NCT_APOL:
00362         APOLparm_dtor(&(calc->apolparm));
00363         break;
00364     default:
00365         Vnm_print(2, "NOsh_calc_ctor: unknown calculation type (%d)!\n",
00366             calc->calctype);
00367         VASSERT(0);
00368 }
00369 PBparm_dtor(&(calc->pbeparm));
00370
00371 Vmem_free(VNULL, 1, sizeof(NOsh_calc), (void **)thee);
00372 calc = VNULL;
00373
00374 }
00375
00376 VPUBLIC int NOsh_calc_copy(
00377     NOsh_calc *thee,
00378     NOsh_calc *source
00379 ) {
00380
00381 VASSERT(thee != VNULL);
00382 VASSERT(source != VNULL);
00383 VASSERT(thee->calctype == source->calctype);
00384 if (source->mgparm != VNULL)
00385     MGparm_copy(thee->mgparm, source->mgparm);
00386 if (source->femparm != VNULL)
00387     FEMparm_copy(thee->femparm, source->femparm);
00388 if (source->pbeparm != VNULL)
00389     PBparm_copy(thee->pbeparm, source->pbeparm);
00390 if (source->apolparm != VNULL)
00391     APOLparm_copy(thee->apolparm, source->apolparm);
00392
00393 return 1;
00394

```

```
00395 }
00396
00397 VPUBLIC int NOsh_parseInputFile(
00398     NOsh *thee,
00399     char *filename
00400 ) {
00401
00402 Vio *sock;
00403 int rc;
00404
00405 sock = Vio_ctor("FILE", "ASC", VNULL, filename, "r");
00406 rc = NOsh_parseInput(thee, sock);
00407 Vio_dtor(&sock);
00408
00409 return rc;
00410 }
00411
00412 VPUBLIC int NOsh_parseInput(
00413     NOsh *thee,
00414     Vio *sock
00415 ) {
00416
00417 char *MCwhiteChars = " ,;\t\r\n";
00418 char *MCcommChars = "%";
00419 char tok[VMAX_BUFSIZE];
00420
00421 if (thee == VNULL) {
00422 Vnm_print(2, "NOsh_parseInput: Got NULL thee!\n");
00423 return 0;
00424 }
00425
00426 if (sock == VNULL) {
00427 Vnm_print(2, "NOsh_parseInput: Got pointer to NULL socket!\n");
00428 Vnm_print(2, "NOsh_parseInput: The specified input file was not found!\n");
00429 return 0;
00430 }
00431
00432 if (thee->parsed) {
00433 Vnm_print(2, "NOsh_parseInput: Already parsed an input file!\n");
00434 return 0;
00435 }
00436
00437 if (Vio_accept(sock, 0) < 0) {
00438 Vnm_print(2, "NOsh_parseInput: Problem reading from socket!\n");
00439 return 0;
00440 }
00441
00442 /* Set up the whitespace and comment character definitions */
00443 Vio_setWhiteChars(sock, MCwhiteChars);
00444 Vio_setCommChars(sock, MCcommChars);
00445
00446 /* We parse the file until we run out of tokens */
00447 Vnm_print(0, "NOsh_parseInput: Starting file parsing...\n");
00448 while (Vio_scanf(sock, "%s", tok) == 1) {
00449 /* At the highest level, we look for keywords that indicate functions like:
00450     read => Read in a molecule file
```

```

00452     elec => Do an electrostatics calculation
00453     print => Print some results
00454     apolar => do a non-polar calculation
00455     quit => Quit
00456
00457     These cause the code to go to a lower-level parser routine which
00458     handles keywords specific to the particular function. Each
00459     lower-level parser routine then returns when it hits the "end"
00460     keyword. Due to this simple layout, no nesting of these "function"
00461     sections is allowed.
00462     */
00463     if (Vstring_strcasecmp(tok, "read") == 0) {
00464         Vnm_print(0, "NOsh: Parsing READ section\n");
00465         if (!NOsh_parseREAD(thee, sock)) return 0;
00466         Vnm_print(0, "NOsh: Done parsing READ section \
00467         (nmol=%d, ndiel=%d, nkappa=%d, ncharge=%d, npot=%d)\n", thee->nmol, thee->ndiel,
00468             thee->nkappa, thee->ncharge, thee->npot);
00469     } else if (Vstring_strcasecmp(tok, "print") == 0) {
00470         Vnm_print(0, "NOsh: Parsing PRINT section\n");
00471         if (!NOsh_parsePRINT(thee, sock)) return 0;
00472         Vnm_print(0, "NOsh: Done parsing PRINT section\n");
00473     } else if (Vstring_strcasecmp(tok, "elec") == 0) {
00474         Vnm_print(0, "NOsh: Parsing ELEC section\n");
00475         if (!NOsh_parseELEC(thee, sock)) return 0;
00476         Vnm_print(0, "NOsh: Done parsing ELEC section (nelec = %d)\n",
00477             thee->nelec);
00478     } else if (Vstring_strcasecmp(tok, "apolar") == 0) {
00479         Vnm_print(0, "NOsh: Parsing APOLAR section\n");
00480         if (!NOsh_parseAPOLAR(thee, sock)) return 0;
00481         Vnm_print(0, "NOsh: Done parsing APOLAR section (nelec = %d)\n",
00482             thee->nelec);
00483     } else if (Vstring_strcasecmp(tok, "quit") == 0) {
00484         Vnm_print(0, "NOsh: Done parsing file (got QUIT)\n");
00485         break;
00486     } else {
00487         Vnm_print(2, "NOsh_parseInput: Ignoring undefined keyword %s!\n", tok);
00488     }
00489 }
00490
00491     thee->parsed = 1;
00492     return 1;
00493
00494 }
00495
00496 VPRIVATE int NOsh_parseREAD_MOL(NOsh *thee, Vio *sock) {
00497
00498     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
00499     NOsh_MolFormat molfmt;
00500
00501     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00502     if (Vstring_strcasecmp(tok, "pqr") == 0) {
00503         molfmt = NMF_PQR;
00504         VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00505         if (tok[0]=='"') {
00506             strcpy(strnew, "");
00507             while (tok[strlen(tok)-1] != '"') {

```

```

00508             strcat(str, tok);
00509             strcat(str, " ");
00510             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00511     }
00512     strcat(str, tok);
00513     strncpy(strnew, str+1, strlen(str)-2);
00514     strcpy(tok, strnew);
00515 }
Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00517     thee->nmol, tok);
00518     thee->molfmt [thee->nmol] = molfmt;
00519     strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00520     (thee->nmol)++;
00521 } else if (Vstring_strcasecmp(tok, "pdb") == 0) {
00522     molfmt = NMF_PDB;
00523     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00524     if (tok[0]== '/') {
00525         strcpy(strnew, "");
00526         while (tok[strlen(tok)-1] != '/') {
00527             strcat(str, tok);
00528             strcat(str, " ");
00529             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00530         }
00531         strcat(str, tok);
00532         strncpy(strnew, str+1, strlen(str)-2);
00533         strcpy(tok, strnew);
00534     }
00535     Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00536     thee->nmol, tok);
00537     thee->molfmt [thee->nmol] = molfmt;
00538     strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00539     (thee->nmol)++;
00540 } else if (Vstring_strcasecmp(tok, "xml") == 0) {
00541     molfmt = NMF_XML;
00542     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00543     if (tok[0]== '/') {
00544         strcpy(strnew, "");
00545         while (tok[strlen(tok)-1] != '/') {
00546             strcat(str, tok);
00547             strcat(str, " ");
00548             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00549         }
00550         strcat(str, tok);
00551         strncpy(strnew, str+1, strlen(str)-2);
00552         strcpy(tok, strnew);
00553     }
00554     Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00555     thee->nmol, tok);
00556     thee->molfmt [thee->nmol] = molfmt;
00557     strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00558     (thee->nmol)++;
00559 } else {
00560     Vnm_print(2, "NOsh_parseREAD: Ignoring undefined mol format \
00561 %s!\n", tok);
00562 }
00563
00564 return 1;

```

```

00565
00566
00567 VERROR1:
00568     Vnm_print(2, "NOsh_parseREAD_MOL: Ran out of tokens while parsing READ s
00569     ection!\n");
00570     return 0;
00571 }
00572
00573 VPRIVATE int NOsh_parseREAD_PARM(NOsh *thee, Vio *sock) {
00574
00575     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00576     NOsh_ParmFormat parmfmt;
00577
00578     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00579     if (Vstring_strcasecmp(tok, "flat") == 0) {
00580         parmfmt = NPF_FLAT;
00581         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00582         if (tok[0]=='"') {
00583             strcpy(strnew, "");
00584             while (tok[strlen(tok)-1] != '"') {
00585                 strcat(str, tok);
00586                 strcat(str, " ");
00587                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00588             }
00589             strcat(str, tok);
00590             strncpy(strnew, str+1, strlen(str)-2);
00591             strcpy(tok, strnew);
00592         }
00593         if (thee->gotparm) {
00594             Vnm_print(2, "NOsh: Hey! You already specified a parameterfile (%s)
00595             !\n", thee->parmpath);
00596             Vnm_print(2, "NOsh: I'm going to ignore this one (%s)!\n", tok);
00597         } else {
00598             thee->parmfmt = parmfmt;
00599             thee->gotparm = 1;
00600             strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00601         }
00602     } else if(Vstring_strcasecmp(tok, "xml") == 0) {
00603         parmfmt = NPF_XML;
00604         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00605         if (tok[0]=='"') {
00606             strcpy(strnew, "");
00607             while (tok[strlen(tok)-1] != '"') {
00608                 strcat(str, tok);
00609                 strcat(str, " ");
00610                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00611             }
00612             strcat(str, tok);
00613             strncpy(strnew, str+1, strlen(str)-2);
00614             strcpy(tok, strnew);
00615         }
00616         if (thee->gotparm) {
00617             Vnm_print(2, "NOsh: Hey! You already specified a parameterfile (%s)
00618             !\n", thee->parmpath);
00619             Vnm_print(2, "NOsh: I'm going to ignore this one (%s)!\n", tok);
00620         } else {

```

```

00619         thee->parmfmt = parmfmt;
00620         thee->gotparm = 1;
00621         strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00622     }
00623
00624 } else {
00625     Vnm_print(2, "NOsh_parseREAD: Ignoring undefined parm format \
00626 %s!\n", tok);
00627 }
00628
00629     return 1;
00630
00631 VERROR1:
00632     Vnm_print(2, "NOsh_parseREAD_PARM: Ran out of tokens while parsing READ
00633 section!\n");
00634     return 0;
00635 }
00636
00637 VPRIVATE int NOsh_parseREAD_DIEL(NOsh *thee, Vio *sock) {
00638
00639     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
00640     Vdata_Format dielfmt;
00641
00642     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00643     if (Vstring_strcasecmp(tok, "dx") == 0) {
00644         dielfmt = VDF_DX;
00645     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00646         dielfmt = VDF_GZ;
00647     } else {
00648         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined format \
00649 %s!\n", tok);
00650         return VRC_FAILURE;
00651     }
00652
00653     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00654     if (tok[0]==')') {
00655         strcpy(strnew, "");
00656         while (tok[strlen(tok)-1] != ')') {
00657             strcat(str, tok);
00658             strcat(str, " ");
00659             VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00660         }
00661         strcat(str, tok);
00662         strncpy(strnew, str+1, strlen(str)-2);
00663         strcpy(tok, strnew);
00664     }
00665     Vnm_print(0, "NOsh: Storing x-shifted dielectric map %d path \
00666 %s\n", thee->ndiel, tok);
00667     strncpy(thee->dielXpath[thee->ndiel], tok, VMAX_ARGLEN);
00668     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00669     Vnm_print(0, "NOsh: Storing y-shifted dielectric map %d path \
00670 %s\n", thee->ndiel, tok);
00671     strncpy(thee->dielYpath[thee->ndiel], tok, VMAX_ARGLEN);
00672     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00673     Vnm_print(0, "NOsh: Storing z-shifted dielectric map %d path \
00674 %s\n", thee->ndiel, tok);

```

```

00675  strncpy(thee->dielZpath[thee->n디엘], tok, VMAX_ARGLEN);
00676  thee->dielfmt[thee->n디엘] = dielfmt;
00677  (thee->n디엘)++;
00678
00679  return 1;
00680
00681 VERROR1:
00682     Vnm_print(2, "NOsh_parseREAD_DIEL: Ran out of tokens while parsing READ \
00683 section!\n");
00684  return 0;
00685
00686 }
00687
00688 VPRIVATE int NOsh_parseREAD_KAPPA(NOsh *thee, Vio *sock) {
00689
00690     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
00691     Vdata_Format kappafmt;
00692
00693     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00694     if (Vstring_strcasecmp(tok, "dx") == 0) {
00695         kappafmt = VDF_DX;
00696     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00697         kappafmt = VDF_GZ;
00698     } else {
00699         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined format \
00700 %s!\n", tok);
00701     return VRC_FAILURE;
00702 }
00703
00704     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00705     if (tok[0]=="/") {
00706         strcpy(strnew, "");
00707         while (tok[strlen(tok)-1] != '/') {
00708             strcat(str, tok);
00709             strcat(str, " ");
00710             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00711         }
00712         strcat(str, tok);
00713         strncpy(strnew, str+1, strlen(str)-2);
00714         strcpy(tok, strnew);
00715     }
00716     Vnm_print(0, "NOsh: Storing kappa map %d path %s\n",
00717               thee->nκappa, tok);
00718     thee->kappafmt[thee->nκappa] = kappafmt;
00719     strncpy(thee->kappapath[thee->nκappa], tok, VMAX_ARGLEN);
00720     (thee->nκappa)++;
00721
00722     return 1;
00723
00724 VERROR1:
00725     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \
00726 section!\n");
00727  return 0;
00728
00729 }
00730

```

```

00731 VPRIVATE int NOsh_parseREAD_POTENTIAL(NOsh *thee, Vio *sock) {
00732
00733     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
00734     Vdata_Format potfmt;
00735
00736     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00737     if (Vstring_strcasecmp(tok, "dx") == 0) {
00738         potfmt = VDF_DX;
00739     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00740         potfmt = VDF_GZ;
00741     } else {
00742         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined format \
00743             %s!\n", tok);
00744         return VRC_FAILURE;
00745     }
00746
00747     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00748     if (tok[0]=='"') {
00749         strcpy(strnew, "");
00750         while (tok[strlen(tok)-1] != '"') {
00751             strcat(str, tok);
00752             strcat(str, " ");
00753             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00754         }
00755         strcat(str, tok);
00756         strncpy(strnew, str+1, strlen(str)-2);
00757         strcpy(tok, strnew);
00758     }
00759     Vnm_print(0, "NOsh: Storing potential map %d path %s\n",
00760             thee->npot, tok);
00761     thee->potfmt[thee->npot] = potfmt;
00762     strncpy(thee->potpath[thee->npot], tok, VMAX_ARGLEN);
00763     (thee->npot)++;
00764
00765     return 1;
00766
00767 VERROR1:
00768     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \
00769             section!\n");
00770     return 0;
00771
00772 }
00773
00774 VPRIVATE int NOsh_parseREAD_CHARGE(NOsh *thee, Vio *sock) {
00775
00776     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
00777     Vdata_Format chargefmt;
00778
00779     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00780     if (Vstring_strcasecmp(tok, "dx") == 0) {
00781         chargefmt = VDF_DX;
00782     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00783         chargefmt = VDF_GZ;
00784     } else {
00785         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined format \
00786             %s!\n", tok);
00787         return VRC_FAILURE;

```

```

00788     }
00789
00790 VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00791 if (tok[0]=='/') {
00792     strcpy(strnew, "");
00793     while (tok[strlen(tok)-1] != '/') {
00794         strcat(str, tok);
00795         strcat(str, " ");
00796         VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00797     }
00798     strcat(str, tok);
00799     strncpy(strnew, str+1, strlen(str)-2);
00800     strcpy(tok, strnew);
00801 }
00802 Vnm_print(0, "NOsh: Storing charge map %d path %s\n",
00803             thee->ncharge, tok);
00804 thee->chargefmt[thee->ncharge] = chargefmt;
00805 strncpy(thee->chargepath[thee->ncharge], tok, VMAX_ARGLEN);
00806 (thee->ncharge)++;
00807
00808     return 1;
00809
00810 VERROR1:
00811     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \
00812 section!\n");
00813     return 0;
00814
00815 }
00816
00817 VPRIVATE int NOsh_parseREAD_MESH(NOsh *thee, Vio *sock) {
00818
00819     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
00820     Vdata_Format meshfmt;
00821
00822     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00823     if (Vstring_strcmp(tok, "mcsf") == 0) {
00824         meshfmt = VDF_MCSF;
00825         VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00826         if (tok[0]=='/') {
00827             strcpy(strnew, "");
00828             while (tok[strlen(tok)-1] != '/') {
00829                 strcat(str, tok);
00830                 strcat(str, " ");
00831                 VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00832             }
00833             strcat(str, tok);
00834             strncpy(strnew, str+1, strlen(str)-2);
00835             strcpy(tok, strnew);
00836         }
00837         Vnm_print(0, "NOsh: Storing mesh %d path %s\n",
00838             thee->nmesh, tok);
00839         thee->meshfmt[thee->nmesh] = meshfmt;
00840         strncpy(thee->meshpath[thee->nmesh], tok, VMAX_ARGLEN);
00841         (thee->nmesh)++;
00842     } else {
00843         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined mesh format \
00844             %s!\n", tok);

```

```

00845     }
00846
00847     return 1;
00848
00849 VERROR1:
00850     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \
00851             section!\n");
00852     return 0;
00853
00854 }
00855
00856
00857 VPRIVATE int NOsh_parseREAD(NOsh *thee, Vio *sock) {
00858
00859     char tok[VMAX_BUFSIZE];
00860
00861     if (thee == VNULL) {
00862         Vnm_print(2, "NOsh_parseREAD: Got NULL thee!\n");
00863         return 0;
00864     }
00865
00866     if (sock == VNULL) {
00867         Vnm_print(2, "NOsh_parseREAD: Got pointer to NULL socket!\n");
00868         return 0;
00869     }
00870
00871     if (thee->parsed) {
00872         Vnm_print(2, "NOsh_parseREAD: Already parsed an input file!\n");
00873         return 0;
00874     }
00875
00876     /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
00877     while (Vio_scanf(sock, "%s", tok) == 1) {
00878         if (Vstring_strcasecmp(tok, "end") == 0) {
00879             Vnm_print(0, "NOsh: Done parsing READ section\n");
00880             return 1;
00881         } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00882             NOsh_parseREAD_MOL(thee, sock);
00883         } else if (Vstring_strcasecmp(tok, "parm") == 0) {
00884             NOsh_parseREAD_PARM(thee, sock);
00885         } else if (Vstring_strcasecmp(tok, "diel") == 0) {
00886             NOsh_parseREAD_DIEL(thee, sock);
00887         } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
00888             NOsh_parseREAD_KAPPA(thee, sock);
00889         } else if (Vstring_strcasecmp(tok, "pot") == 0) {
00890             NOsh_parseREAD_POTENTIAL(thee, sock);
00891         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00892             NOsh_parseREAD_CHARGE(thee, sock);
00893         } else if (Vstring_strcasecmp(tok, "mesh") == 0) {
00894             NOsh_parseREAD_MESH(thee, sock);
00895         } else {
00896             Vnm_print(2, "NOsh_parseREAD: Ignoring undefined keyword %s!\n",
00897                     tok);
00898         }
00899     }
00900
00901     /* We ran out of tokens! */

```

```

00902     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \
00903 section!\n");
00904     return 0;
00905
00906 }
00907
00908 VPRIVATE int NOsh_parsePRINT(NOsh *thee, Vio *sock) {
00909
00910     char tok[VMAX_BUFSIZE];
00911     char name[VMAX_BUFSIZE];
00912     int ti, idx, expect, ielec, iapol;
00913
00914     if (thee == VNULL) {
00915         Vnm_print(2, "NOsh_parsePRINT: Got NULL thee!\n");
00916         return 0;
00917     }
00918
00919     if (sock == VNULL) {
00920         Vnm_print(2, "NOsh_parsePRINT: Got pointer to NULL socket!\n");
00921         return 0;
00922     }
00923
00924     if (thee->parsed) {
00925         Vnm_print(2, "NOsh_parsePRINT: Already parsed an input file!\n");
00926         return 0;
00927     }
00928
00929     idx = thee->nprint;
00930     if (thee->nprint >= NOSH_MAXPRINT) {
00931         Vnm_print(2, "NOsh_parsePRINT: Exceeded max number (%d) of PRINT \
00932 sections\n",
00933             NOSH_MAXPRINT);
00934         return 0;
00935     }
00936
00937
00938     /* The first thing we read is the thing we want to print */
00939     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00940     if (Vstring_strcasecmp(tok, "energy") == 0) {
00941         thee->printwhat[idx] = NPT_ENERGY;
00942         thee->printnarg[idx] = 0;
00943     } else if (Vstring_strcasecmp(tok, "force") == 0) {
00944         thee->printwhat[idx] = NPT_FORCE;
00945         thee->printnarg[idx] = 0;
00946     } else if (Vstring_strcasecmp(tok, "elecEnergy") == 0) {
00947         thee->printwhat[idx] = NPT_ELECENERGY;
00948         thee->printnarg[idx] = 0;
00949     } else if (Vstring_strcasecmp(tok, "elecForce") == 0) {
00950         thee->printwhat[idx] = NPT_ELECFORCE;
00951         thee->printnarg[idx] = 0;
00952     } else if (Vstring_strcasecmp(tok, "apolEnergy") == 0) {
00953         thee->printwhat[idx] = NPT_APOLENERGY;
00954         thee->printnarg[idx] = 0;
00955     } else if (Vstring_strcasecmp(tok, "apolForce") == 0) {
00956         thee->printwhat[idx] = NPT_APOLFORCE;
00957         thee->printnarg[idx] = 0;
00958     } else {

```

```

00959         Vnm_print(2, "NOsh_parsePRINT: Undefined keyword %s while parsing \
00960 PRINT section!\n", tok);
00961         return 0;
00962     }
00963
00964     expect = 0; /* We first expect a calculation ID (0) then an op (1) */
00965
00966 /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
00967 while (Vio_scanf(sock, "%s", tok) == 1) {
00968
00969     /* The next thing we read is either END or an ARG OP ARG statement */
00970     if (Vstring_strcasecmp(tok, "end") == 0) {
00971         if (expect != 0) {
00972             (thee->nprint)++;
00973             (thee->printnarg[idx])++;
00974             Vnm_print(0, "NOsh: Done parsing PRINT section\n");
00975             return 1;
00976         } else {
00977             Vnm_print(2, "NOsh_parsePRINT: Got premature END to PRINT!\n");
00978             return 0;
00979         }
00980     } else {
00981
00982         /* Grab a calculation ID */
00983         if ((sscanf(tok, "%d", &ti) == 1) &&
00984 (Vstring_isdigit(tok) == 1)) {
00985             if (expect == 0) {
00986                 thee->printcalc[idx][thee->printnarg[idx]] = ti-1;
00987                 expect = 1;
00988             } else {
00989                 Vnm_print(2, "NOsh_parsePRINT: Syntax error in PRINT \
00990 section while reading %s!\n", tok);
00991                 return 0;
00992             }
00993         /* Grab addition operation */
00994         } else if (Vstring_strcasecmp(tok, "+") == 0) {
00995             if (expect == 1) {
00996                 thee->printop[idx][thee->printnarg[idx]] = 0;
00997                 (thee->printnarg[idx])++;
00998                 expect = 0;
00999                 if (thee->printnarg[idx] >= NOSH_MAXPOP) {
01000                     Vnm_print(2, "NOsh_parsePRINT: Exceeded max number \
01001 (%d) of arguments for PRINT section!\n",
01002                     NOSH_MAXPOP);
01003                     return 0;
01004                 }
01005             } else {
01006                 Vnm_print(2, "NOsh_parsePRINT: Syntax error in PRINT \
01007 section while reading %s!\n", tok);
01008                 return 0;
01009             }
01010         /* Grab subtraction operation */
01011         } else if (Vstring_strcasecmp(tok, "-") == 0) {
01012             if (expect == 1) {
01013                 thee->printop[idx][thee->printnarg[idx]] = 1;
01014                 (thee->printnarg[idx])++;
01015                 expect = 0;

```

```

01016             if (thee->printnarg[idx] >= NOSH_MAXPOP) {
01017                 Vnm_print(2, "Nosh_parseREAD: Exceeded max number \
01018 (%d) of arguments for PRINT section!\n",
01019                         NOSH_MAXPOP);
01020                 return 0;
01021             }
01022         } else {
01023             Vnm_print(2, "Nosh_parsePRINT: Syntax error in PRINT \
01024 section while reading %s!\n", tok);
01025             return 0;
01026         }
01027     /* Grab a calculation name from elec ID */
01028 } else if (sscanf(tok, "%s", name) == 1) {
01029     if (expect == 0) {
01030         for (ielec=0; ielec<thee->nelec; ielec++) {
01031             if (Vstring_strcasecmp(thee->elecname[ielec], name) == 0) {
01032                 thee->printcalc[idx][thee->printnarg[idx]] = ielec;
01033                 expect = 1;
01034                 break;
01035             }
01036         }
01037         for (iapol=0; iapol<thee->nopol; iapol++) {
01038             if (Vstring_strcasecmp(thee->apolname[iapol], name) == 0) {
01039                 thee->printcalc[idx][thee->printnarg[idx]] = iapol;
01040                 expect = 1;
01041                 break;
01042             }
01043         }
01044     if (expect == 0) {
01045         Vnm_print(2, "No ELEC or APOL statement has been named %s!\n",
01046                   name);
01047         return 0;
01048     }
01049     } else {
01050         Vnm_print(2, "Nosh_parsePRINT: Syntax error in PRINT \
01051 section while reading %s!\n", tok);
01052         return 0;
01053     }
01054 /* Got bad operation */
01055 } else {
01056     Vnm_print(2, "Nosh_parsePRINT: Undefined keyword %s while \
01057 parsing PRINT section!\n", tok);
01058     return 0;
01059 }
01060 } /* end parse token */
01061
01062 } /* end while */
01063
01064 VJMPERR1(0);
01065
01066 /* We ran out of tokens! */
01067 VERROR1:
01068     Vnm_print(2, "Nosh_parsePRINT: Ran out of tokens while parsing PRINT \
01069 section!\n");
01070     return 0;
01071
01072 }

```

```

01073
01074 VPRIVATE int NOsh_parseELEC(NOsh *thee, Vio *sock) {
01075
01076     NOsh_calc *calc = VNULL;
01077
01078     char tok[VMAX_BUFSIZE];
01079
01080     if (thee == VNULL) {
01081         Vnm_print(2, "NOsh_parseELEC: Got NULL thee!\n");
01082         return 0;
01083     }
01084
01085     if (sock == VNULL) {
01086         Vnm_print(2, "NOsh_parseELEC: Got pointer to NULL socket!\n");
01087         return 0;
01088     }
01089
01090     if (thee->parsed) {
01091         Vnm_print(2, "NOsh_parseELEC: Already parsed an input file!\n");
01092         return 0;
01093     }
01094
01095     /* Get a pointer to the latest ELEC calc object and update the ELEC
01096 statement number */
01097     if (thee->nelec >= NOSH_MAXCALC) {
01098         Vnm_print(2, "NOsh: Too many electrostatics calculations in this \
01099 run!\n");
01100         Vnm_print(2, "NOsh: Current max is %d; ignoring this calculation\n",
01101             NOSH_MAXCALC);
01102         return 1;
01103     }
01104
01105     /* The next token HAS to be the method OR "name" */
01106     if (Vio_scanf(sock, "%s", tok) == 1) {
01107         if (Vstring_strcasecmp(tok, "name") == 0) {
01108             Vio_scanf(sock, "%s", tok);
01109             strncpy(thee->elecname[thee->nelec], tok, VMAX_ARGLEN);
01110             if (Vio_scanf(sock, "%s", tok) != 1) {
01111                 Vnm_print(2, "NOsh_parseELEC: Ran out of tokens while reading \
01112 ELEC section!\n");
01113                 return 0;
01114             }
01115         }
01116         if (Vstring_strcasecmp(tok, "mg-manual") == 0) {
01117             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01118             calc = thee->elec[thee->nelec];
01119             (thee->nelec)++;
01120             calc->mparm->type = MCT_MANUAL;
01121             return NOsh_parseMG(thee, sock, calc);
01122         } else if (Vstring_strcasecmp(tok, "mg-auto") == 0) {
01123             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01124             calc = thee->elec[thee->nelec];
01125             (thee->nelec)++;
01126             calc->mparm->type = MCT_AUTO;
01127             return NOsh_parseMG(thee, sock, calc);
01128         } else if (Vstring_strcasecmp(tok, "mg-para") == 0) {
01129             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);

```

```

01130     calc = theee->elec[theee->nelec];
01131     (theee->nelec)++;
01132     calc->mparm->type = MCT_PARALLEL;
01133     return NOsh_parseMG(theee, sock, calc);
01134 } else if (Vstring_strcasecmp(tok, "mg-dummy") == 0) {
01135     theee->elec[theee->nelec] = NOSH_calc_ctor(NCT_MG);
01136     calc = theee->elec[theee->nelec];
01137     (theee->nelec)++;
01138     calc->mparm->type = MCT_DUMMY;
01139     return NOsh_parseMG(theee, sock, calc);
01140 } else if (Vstring_strcasecmp(tok, "fe-manual") == 0) {
01141     theee->elec[theee->nelec] = NOSH_calc_ctor(NCT_FEM);
01142     calc = theee->elec[theee->nelec];
01143     (theee->nelec)++;
01144     calc->femparm->type = FCT_MANUAL;
01145     return NOsh_parseFEM(theee, sock, calc);
01146 } else {
01147     Vnm_print(2, "NOsh_parseELEC: The method (\\"mg\\" or \\"fem\\") or \
01148 \\"name\\" must be the first keyword in the ELEC section\n");
01149     return 0;
01150 }
01151 }
01152
01153     Vnm_print(2, "NOsh_parseELEC: Ran out of tokens while reading ELEC section!\\
n");
01154     return 0;
01155
01156 }
01157
01158 VPRIVATE int NOsh_parseAPOLAR(NOsh *thee, Vio *sock) {
01159
01160     NOsh_calc *calc = VNULL;
01161
01162     char tok[VMAX_BUFSIZE];
01163
01164     if (thee == VNULL) {
01165         Vnm_print(2, "NOsh_parseAPOLAR: Got NULL thee!\n");
01166         return 0;
01167     }
01168
01169     if (sock == VNULL) {
01170         Vnm_print(2, "NOsh_parseAPOLAR: Got pointer to NULL socket!\n");
01171         return 0;
01172     }
01173
01174     if (thee->parsed) {
01175         Vnm_print(2, "NOsh_parseAPOLAR: Already parsed an input file!\n");
01176         return 0;
01177     }
01178
01179 /* Get a pointer to the latest ELEC calc object and update the ELEC
01180 statement number */
01181     if (thee->nopol >= NOSH_MAXCALC) {
01182         Vnm_print(2, "NOsh: Too many non-polar calculations in this \
01183 run!\n");
01184         Vnm_print(2, "NOsh: Current max is %d; ignoring this calculation\n",
01185         NOSH_MAXCALC);

```

```

01186         return 1;
01187     }
01188
01189 /* The next token HAS to be the method OR "name" */
01190 if (Vio_scanf(sock, "%s", tok) == 1) {
01191     if (Vstring_strcasecmp(tok, "name") == 0) {
01192         Vio_scanf(sock, "%s", tok);
01193         strncpy(thee->apolname[thee->napol], tok, VMAX_ARGLEN);
01194
01195 /* Parse the non-polar parameters */
01196 thee->apol[thee->napol] = NOsh_calc_ctor(NCT_APOL);
01197 calc = thee->apol[thee->napol];
01198 (thee->napol)++;
01199 return NOsh_parseAPOL(thee, sock, calc);
01200
01201 if (Vio_scanf(sock, "%s", tok) != 1) {
01202     Vnm_print(2, "NOsh_parseAPOL: Ran out of tokens while reading \
01203 APOLAR section!\n");
01204     return 0;
01205 }
01206 }
01207 }
01208
01209 return 1;
01210
01211 }
01212
01213 VPUBLIC int NOsh_setupElecCalc(
01214     NOsh *thee,
01215     Valist *alist[NOSH_MAXMOL]
01216     ) {
01217 int ielec, imol, i;
01218 NOsh_calc *elec = VNULL;
01219 MGparm *mgparm = VNULL;
01220 Valist *mymol = VNULL;
01221
01222 VASSERT(thee != VNULL);
01223 for (imol=0; imol<thee->nmol; imol++) {
01224     thee->alist[imol] = alist[imol];
01225 }
01226
01227
01228 for (ielec=0; ielec<(thee->nelec); ielec++) {
01229 /* Unload the calculation object containing the ELEC information */
01230     elec = thee->elec[ielec];
01231
01232 if (((thee->nndiel != 0) || (thee->nkappa != 0) ||
01233 (thee->ncharge != 0) || (thee->npot != 0)) &&
01234 (elec->pbeparm->calctype != PCF_NO)) {
01235     Vnm_print(2, "NOsh_setupElecCalc: Calculation of forces disabled because surf
ace \
01236 map is used!\n");
01237     elec->pbeparm->calctype = PCF_NO;
01238 }
01239
01240 /* Setup the calculation */
01241 switch (elec->calctype) {

```

```

01242     case NCT_MG:
01243         /* Center on the molecules, if requested */
01244         mgparm = elec->mgparm;
01245         VASSERT(mgparm != VNULL);
01246         if (elec->mgparm->cmeth == MCM_MOLECULE) {
01247             VASSERT(mgparm->centmol >= 0);
01248             VASSERT(mgparm->centmol < thee->nmol);
01249             mymol = thee->alist[mgparm->centmol];
01250             VASSERT(mymol != VNULL);
01251             for (i=0; i<3; i++) {
01252                 mgparm->center[i] = mymol->center[i];
01253             }
01254         }
01255         if (elec->mgparm->fcmeth == MCM_MOLECULE) {
01256             VASSERT(mgparm->fcentmol >= 0);
01257             VASSERT(mgparm->fcentmol < thee->nmol);
01258             mymol = thee->alist[mgparm->fcentmol];
01259             VASSERT(mymol != VNULL);
01260             for (i=0; i<3; i++) {
01261                 mgparm->fcenter[i] = mymol->center[i];
01262             }
01263         }
01264         if (elec->mgparm->ccmeth == MCM_MOLECULE) {
01265             VASSERT(mgparm->ccentmol >= 0);
01266             VASSERT(mgparm->ccentmol < thee->nmol);
01267             mymol = thee->alist[mgparm->ccentmol];
01268             VASSERT(mymol != VNULL);
01269             for (i=0; i<3; i++) {
01270                 mgparm->ccenter[i] = mymol->center[i];
01271             }
01272         }
01273         NOsh_setupCalcMG(thee, elec);
01274         break;
01275     case NCT_FEM:
01276         NOsh_setupCalcFEM(thee, elec);
01277         break;
01278     default:
01279         Vnm_print(2, "NOsh_setupCalc: Invalid calculation type (%d)!\n",
01280                   elec->calctype);
01281         return 0;
01282     }
01283
01284     /* At this point, the most recently-created NOsh_calc object should be the
01285      one we use for results for this ELEC statement. Assign it. */
01286     /* Associate ELEC statement with the calculation */
01287     thee->elec2calc[ielec] = thee->ncalc-1;
01288     Vnm_print(0, "NOsh_setupCalc: Mapping ELEC statement %d (%d) to \
01289 calculation %d (%d)\n", ielec, ielec+1, thee->elec2calc[ielec],
01290               thee->elec2calc[ielec]+1);
01291   }
01292
01293   return 1;
01294 }
01295
01296 VPUBLIC int NOsh_setupApolCalc(
01297           NOsh *thee,
01298           Valist *alist[NOSH_MAXMOL]

```

```

01299         ) {
01300     int iapol, imol;
01301     int doCalc = ACD_NO;
01302     NOsh_calc *calc = VNULL;
01303
01304     VASSERT(thee != VNULL);
01305     for (imol=0; imol<thee->nmol; imol++) {
01306         thee->alist[imol] = alist[imol];
01307     }
01308
01309     for (iapol=0; iapol<(thee->nopol); iapol++) {
01310         /* Unload the calculation object containing the APOL information */
01311         calc = thee->apol[iapol];
01312
01313         /* Setup the calculation */
01314         switch (calc->calctype) {
01315             case NCT_APOL:
01316                 NOsh_setupCalcAPOL(thee, calc);
01317                 doCalc = ACD_YES;
01318                 break;
01319             default:
01320                 Vnm_print(2, "NOsh_setupCalc: Invalid calculation type (%d)!\n", calc->
01321                           calctype);
01322                 return ACD_ERROR;
01323         }
01324         /* At this point, the most recently-created NOsh_calc object should be the
01325            one we use for results for this APOL statement. Assign it. */
01326         /* Associate APOL statement with the calculation */
01327         thee->apol2calc[iapol] = thee->nCalc-1;
01328         Vnm_print(0, "NOsh_setupCalc: Mapping APOL statement %d (%d) to calculation %d
01329             (%d)\n", iapol, iapol+1, thee->apol2calc[iapol], thee->apol2calc[iapol]+1);
01330     }
01331
01332     if (doCalc == ACD_YES) {
01333         return ACD_YES;
01334     } else{
01335         return ACD_NO;
01336     }
01337 VPUBLIC int NOsh_parseMG(
01338     NOsh *thee,
01339     Vio *sock,
01340     NOsh_calc *elec
01341     ) {
01342
01343     char tok[VMAX_BUFSIZE];
01344     MGparm *mgparm = VNULL;
01345     PBEparm *pbeparm = VNULL;
01346     int rc;
01347
01348     /* Check the arguments */
01349     if (thee == VNULL) {
01350         Vnm_print(2, "NOsh: Got NULL thee!\n");
01351         return 0;
01352     }
01353     if (sock == VNULL) {

```

```

01354     Vnm_print(2, "NOsh: Got pointer to NULL socket!\n");
01355     return 0;
01356 }
01357 if (elec == VNULL) {
01358     Vnm_print(2, "NOsh: Got pointer to NULL elec object!\n");
01359     return 0;
01360 }
01361 mgparm = elec->mgparm;
01362 if (mgparm == VNULL) {
01363     Vnm_print(2, "NOsh: Got pointer to NULL mgparm object!\n");
01364     return 0;
01365 }
01366 pbeparm = elec->pbeparm;
01367 if (pbeparm == VNULL) {
01368     Vnm_print(2, "NOsh: Got pointer to NULL pbeparm object!\n");
01369     return 0;
01370 }
01371
01372 Vnm_print(0, "NOsh_parseMG: Parsing parameters for MG calculation\n");
01373
/* Parallel stuff */
01375 if (mgparm->type == MCT_PARALLEL) {
01376     mgparm->proc_rank = thee->proc_rank;
01377     mgparm->proc_size = thee->proc_size;
01378     mgparm->setrank = 1;
01379     mgparm->setszie = 1;
01380 }
01381
01382 /* Start snarfing tokens from the input stream */
01384 rc = 1;
01385 while (Vio_scanf(sock, "%s", tok) == 1) {
01386
01387     Vnm_print(0, "NOsh_parseMG: Parsing %s...\n", tok);
01388
/* See if it's an END token */
01390 if (Vstring_strcasecmp(tok, "end") == 0) {
01391     mgparm->parsed = 1;
01392     pbeparm->parsed = 1;
01393     rc = 1;
01394     break;
01395 }
01396
/* Pass the token through a series of parsers */
01397 rc = PBEparam_parseToken(pbeparm, tok, sock);
01399 if (rc == -1) {
01400     Vnm_print(0, "NOsh_parseMG: parsePBE error!\n");
01401     break;
01402 } else if (rc == 0) {
/* Pass the token to the generic MG parser */
01404 rc = MGparm_parseToken(mgparm, tok, sock);
01405 if (rc == -1) {
01406     Vnm_print(0, "NOsh_parseMG: parseMG error!\n");
01407     break;
01408 } else if (rc == 0) {
/* We ran out of parsers! */
01409     Vnm_print(2, "NOsh: Unrecognized keyword: %s\n", tok);
01410 }
```

```

01411     break;
01412 }
01413 }
01414 }
01415
01416 /* Handle various errors arising in the token-snarfing loop -- these all
01417 just result in simple returns right now */
01418 if (rc == -1) return 0;
01419 if (rc == 0) return 0;
01420
01421 /* Check the status of the parameter objects */
01422 if ((MGparm_check(mgparm) == VRC_FAILURE) || (!PBEParm_check(pbeparm))) {
01423     Vnm_print(2, "NOsh: MG parameters not set correctly!\n");
01424     return 0;
01425 }
01426
01427 return 1;
01428 }
01429
01430 VPRIVATE int NOsh_setupCalcMG(
01431     NOsh *thee,
01432     NOsh_calc *calc
01433 ) {
01434
01435     MGparm *mgparm = VNULL;
01436
01437     VASSERT(thee != VNULL);
01438     VASSERT(calc != VNULL);
01439     mgparm = calc->mgparm;
01440     VASSERT(mgparm != VNULL);
01441
01442
01443 /* Now we're ready to whatever sorts of post-processing operations that are
01444 necessary for the various types of calculations */
01445 switch (mgparm->type) {
01446     case MCT_MANUAL:
01447         return NOsh_setupCalcMGMANUAL(thee, calc);
01448     case MCT_DUMMY:
01449         return NOsh_setupCalcMGMANUAL(thee, calc);
01450     case MCT_AUTO:
01451         return NOsh_setupCalcMGAUTO(thee, calc);
01452     case MCT_PARALLEL:
01453         return NOsh_setupCalcMGPARA(thee, calc);
01454     default:
01455         Vnm_print(2, "NOsh_setupCalcMG: undefined MG calculation type (%d)!\n",
01456                 mgparm->type);
01457         return 0;
01458 }
01459
01460 /* Shouldn't get here */
01461 return 0;
01462 }
01463
01464 VPRIVATE int NOsh_setupCalcFEM(
01465     NOsh *thee,
01466     NOsh_calc *calc
01467 ) {

```

```

01468
01469 VASSERT(thee != VNULL);
01470 VASSERT(calc != VNULL);
01471 VASSERT(calc->femparm != VNULL);
01472
01473 /* Now we're ready to whatever sorts of post-processing operations that are
01474 * necessary for the various types of calculations */
01475 switch (calc->femparm->type) {
01476 case FCT_MANUAL:
01477     return NOsh_setupCalcFEMANUAL(thee, calc);
01478 default:
01479     Vnm_print(2, "NOsh_parseFEM: unknown calculation type (%d)!\n",
01480             calc->femparm->type);
01481     return 0;
01482 }
01483
01484 /* Shouldn't get here */
01485 return 0;
01486 }
01487
01488
01489 VPRIVATE int NOsh_setupCalcMGMANUAL(
01490         Nosh *thee,
01491         Nosh_calc *elec
01492     ) {
01493
01494     MGparm *mgparm = VNULL;
01495     PBEparm *pbeparm = VNULL;
01496     Nosh_calc *calc = VNULL;
01497
01498     if (thee == VNULL) {
01499         Vnm_print(2, "NOsh_setupCalcMGMANUAL: Got NULL thee!\n");
01500         return 0;
01501     }
01502     if (elec == VNULL) {
01503         Vnm_print(2, "NOsh_setupCalcMGMANUAL: Got NULL calc!\n");
01504         return 0;
01505     }
01506     mgparm = elec->mgparm;
01507     if (mgparm == VNULL) {
01508         Vnm_print(2, "NOsh_setupCalcMGMANUAL: Got NULL mgparm -- was this calculation
 \
01509 set up?\n");
01510         return 0;
01511     }
01512     pbeparm = elec->pbeparm;
01513     if (pbeparm == VNULL) {
01514         Vnm_print(2, "NOsh_setupCalcMGMANUAL: Got NULL pbeparm -- was this calculation
 \
01515 set up?\n");
01516         return 0;
01517     }
01518
01519 /* Set up missing MG parameters */
01520     if (mgparm->setgrid == 0) {
01521         VASSERT(mgparm->setglen);
01522         mgparm->grid[0] = mgparm->glen[0]/((double)(mgparm->dime[0]-1));

```

```

01523     mgparm->grid[1] = mgparm->glen[1]/((double)(mgparm->dime[1]-1));
01524     mgparm->grid[2] = mgparm->glen[2]/((double)(mgparm->dime[2]-1));
01525 }
01526 if (mgparm->setglen == 0) {
01527     VASERT(mgparm->setgrid);
01528     mgparm->glen[0] = mgparm->grid[0]*((double)(mgparm->dime[0]-1));
01529     mgparm->glen[1] = mgparm->grid[1]*((double)(mgparm->dime[1]-1));
01530     mgparm->glen[2] = mgparm->grid[2]*((double)(mgparm->dime[2]-1));
01531 }
01532
01533 /* Check to see if he have any room left for this type of calculation, if
01534 so: set the calculation type, update the number of calculations of this type,
01535 and parse the rest of the section */
01536 if (thee->ncalc >= NOSH_MAXCALC) {
01537     Vnm_print(2, "NOSH: Too many calculations in this run!\n");
01538     Vnm_print(2, "NOSH: Current max is %d; ignoring this calculation\n",
01539             NOSH_MAXCALC);
01540     return 0;
01541 }
01542
01543 /* Get the next calculation object and increment the number of calculations */
01544
01545 thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_MG);
01546 calc = thee->calc[thee->ncalc];
01547 (thee->ncalc)++;
01548
01549
01550 /* Copy over contents of ELEC */
01551 NOsh_calc_copy(calc, elec);
01552
01553
01554     return 1;
01555 }
01556
01557 VPUBLIC int NOsh_setupCalcMGAUTO(
01558         NOsh *thee,
01559         NOsh_calc *elec
01560     ) {
01561
01562     NOsh_calc *calcf = VNULL;
01563     NOsh_calc *calcc = VNULL;
01564     double fgrid[3], cgrid[3];
01565     double d[3], minf[3], maxf[3], minc[3], maxc[3];
01566     double redfrac, redrat[3], td;
01567     int ifocus, nfocus, tnfocus[3];
01568     int j;
01569     int icalc;
01570     int dofix;
01571
01572 /* A comment about the coding style in this function. I use lots and lots
01573 and lots of pointer deferring. I could (and probably should) save
01574 these in temporary variables. However, since there are so many MGparm,
01575 etc. and NOsh_calc, etc. objects running around in this function, the
01576 current scheme is easiest to debug. */
01577
01578

```

```

01579 if (thee == VNULL) {
01580   Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL thee!\n");
01581   return 0;
01582 }
01583 if (elec == VNULL) {
01584   Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL elec!\n");
01585   return 0;
01586 }
01587 if (elec->mgparm == VNULL) {
01588   Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL mgparm!\n");
01589   return 0;
01590 }
01591 if (elec->pbeparm == VNULL) {
01592   Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL pbeparm!\n");
01593   return 0;
01594 }
01595
01596 Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d): coarse grid center = %g %g %g\n",
01597   __FILE__, __LINE__,
01598   elec->mgparm->ccenter[0],
01599   elec->mgparm->ccenter[1],
01600   elec->mgparm->ccenter[2]);
01601 Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d): fine grid center = %g %g %g\n",
01602   __FILE__, __LINE__,
01603   elec->mgparm->fcenter[0],
01604   elec->mgparm->fcenter[1],
01605   elec->mgparm->fcenter[2]);
01606
01607 /* Calculate the grid spacing on the coarse and fine levels */
01608 for (j=0; j<3; j++) {
01609   cgrid[j] = (elec->mgparm->crlen[j])/((double)(elec->mgparm->dime[j]-1));
01610   fgrid[j] = (elec->mgparm->flen[j])/((double)(elec->mgparm->dime[j]-1));
01611   d[j] = elec->mgparm->fcenter[j] - elec->mgparm->ccenter[j];
01612 }
01613 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): Coarse grid spacing = %g, %g, %g\n",
01614   __FILE__, __LINE__, cgrid[0], cgrid[1], cgrid[2]);
01615 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): Fine grid spacing = %g, %g, %g\n",
01616   __FILE__, __LINE__, fgrid[0], fgrid[1], fgrid[2]);
01617 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): Displacement between fine and \
01618 coarse grids = %g, %g, %g\n", __FILE__, __LINE__, d[0], d[1], d[2]);
01619
01620 /* Now calculate the number of focusing levels, never reducing the grid
01621 spacing by more than redfrac at each level */
01622 for (j=0; j<3; j++) {
01623   if (fgrid[j]/cgrid[j] < VREDFRAC) {
01624     redfrac = fgrid[j]/cgrid[j];
01625     td = log(redfrac)/log(VREDFRAC);
01626     tnfocus[j] = (int)ceil(td) + 1;
01627   } else tnfocus[j] = 2;
01628 }
01629 nfocus = VMAX2(VMAX2(tnfocus[0], tnfocus[1]), tnfocus[2]);
01630
01631 /* Now set redrat to the actual value by which the grid spacing is reduced
01632 at each level of focusing */
01633 for (j=0; j<3; j++) {

```

```

01634     redrat[j] = VPOW((fgrid[j]/cgrid[j]), 1.0/((double)nfocus-1.0));
01635 }
01636 Vnm_print(0, "NOsh: %d levels of focusing with %g, %g, %g reductions\n",
01637     nfocus, redrat[0], redrat[1], redrat[2]);
01638
01639 /* Now that we know how many focusing levels to use, we're ready to set up
01640 the parameter objects */
01641 if (nfocus > (NOSH_MAXCALC-(thee->n calc))) {
01642     Vnm_print(2, "NOsh: Require more calculations than max (%d)!\n",
01643     NOSH_MAXCALC);
01644     return 0;
01645 }
01646
01647 for (ifocus=0; ifocus<nfocus; ifocus++) {
01648
01649 /* Generate the new calc object */
01650 icalc = thee->n calc;
01651 thee->calc[icalc] = NOsh_calc_ctor(NCT_MG);
01652 (thee->n calc)++;
01653
01654 /* This is the _current_ NOsh_calc object */
01655 calcf = thee->calc[icalc];
01656 /* This is the _previous_ Nosh_calc object */
01657 if (ifocus != 0) {
01658     calcc = thee->calc[icalc-1];
01659 } else {
01660     calcc = VNULL;
01661 }
01662
01663 /* Copy over most of the parameters from the ELEC object */
01664 NOsh_calc_copy(calcf, elec);
01665
01666 /* Set up the grid lengths and spacings */
01667 if (ifocus == 0) {
01668     for (j=0; j<3; j++) {
01669         calcf->mgparm->grid[j] = cgrid[j];
01670         calcf->mgparm->glen[j] = elec->mgparm->cglen[j];
01671     }
01672 } else {
01673     for (j=0; j<3; j++) {
01674         calcf->mgparm->grid[j] = redrat[j]*(calcc->mgparm->grid[j]);
01675         calcf->mgparm->glen[j] = redrat[j]*(calcc->mgparm->glen[j]);
01676     }
01677 }
01678 calcf->mgparm->setgrid = 1;
01679 calcf->mgparm->setglen = 1;
01680
01681 /* Get centers and centering method from coarse and fine meshes */
01682 if (ifocus == 0) {
01683     calcf->mgparm->cmeth = elec->mgparm->ccmeth;
01684     calcf->mgparm->centmol = elec->mgparm->ccentmol;
01685     for (j=0; j<3; j++) {
01686         calcf->mgparm->center[j] = elec->mgparm->ccenter[j];
01687     }
01688 } else if (ifocus == (nfocus-1)) {
01689     calcf->mgparm->cmeth = elec->mgparm->fcmeth;
01690     calcf->mgparm->centmol = elec->mgparm->fcenmol;

```

```

01691     for (j=0; j<3; j++) {
01692         calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
01693     }
01694 } else {
01695     calcf->mgparm->cmeth = MCM_FOCUS;
01696 /* TEMPORARILY move the current grid center
01697 to the fine grid center. In general, this will move portions of
01698 the current mesh off the immediately-coarser mesh. We'll fix that
01699 in the next step. */
01700 for (j=0; j<3; j++) {
01701     calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
01702 }
01703 }
01704
01705
01706 /* As mentioned above, it is highly likely that the previous "jump"
01707 to the fine grid center put portions of the current mesh off the
01708 previous (coarser) mesh. Fix this by displacing the current mesh
01709 back onto the previous coarser mesh. */
01710 if (ifocus != 0) {
01711     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): starting mesh \
01712 repositioning.\n", __FILE__, __LINE__);
01713     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): coarse mesh center = \
01714 %g %g %g\n", __FILE__, __LINE__,
01715             calcc->mgparm->center[0],
01716             calcc->mgparm->center[1],
01717             calcc->mgparm->center[2]);
01718     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): coarse mesh upper corner = \
01719 %g %g %g\n", __FILE__, __LINE__,
01720             calcc->mgparm->center[0]+0.5*(calcc->mgparm->glen[0]),
01721             calcc->mgparm->center[1]+0.5*(calcc->mgparm->glen[1]),
01722             calcc->mgparm->center[2]+0.5*(calcc->mgparm->glen[2]));
01723     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): coarse mesh lower corner = \
01724 %g %g %g\n", __FILE__, __LINE__,
01725             calcc->mgparm->center[0]-0.5*(calcc->mgparm->glen[0]),
01726             calcc->mgparm->center[1]-0.5*(calcc->mgparm->glen[1]),
01727             calcc->mgparm->center[2]-0.5*(calcc->mgparm->glen[2]));
01728     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): initial fine mesh upper corner = \
01729 %g %g %g\n", __FILE__, __LINE__,
01730             calcf->mgparm->center[0]+0.5*(calcf->mgparm->glen[0]),
01731             calcf->mgparm->center[1]+0.5*(calcf->mgparm->glen[1]),
01732             calcf->mgparm->center[2]+0.5*(calcf->mgparm->glen[2]));
01733     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): initial fine mesh lower corner = \
01734 %g %g %g\n", __FILE__, __LINE__,
01735             calcf->mgparm->center[0]-0.5*(calcf->mgparm->glen[0]),
01736             calcf->mgparm->center[1]-0.5*(calcf->mgparm->glen[1]),
01737             calcf->mgparm->center[2]-0.5*(calcf->mgparm->glen[2]));
01738 for (j=0; j<3; j++) {
01739     /* Check if we've fallen off of the lower end of the mesh */
01740     dofix = 0;
01741     minf[j] = calcf->mgparm->center[j]
01742             - 0.5*(calcf->mgparm->glen[j]);
01743     minc[j] = calcc->mgparm->center[j]
01744             - 0.5*(calcc->mgparm->glen[j]);
01745     d[j] = minc[j] - minf[j];

```

```

01746     if (d[j] >= VSMALL) {
01747         if (ifocus == (nfocus-1)) {
01748             Vnm_print(2, "NOsh_setupCalcMGAUTO: Error! Finest \
01749 mesh has fallen off the coarser meshes!\n");
01750             Vnm_print(2, "NOsh_setupCalcMGAUTO: difference in min %d-\
01751 direction = %g\n", j, d[j]);
01752             Vnm_print(2, "NOsh_setupCalcMGAUTO: min fine = %g %g %g\n",
01753                     minf[0], minf[1], minf[2]);
01754             Vnm_print(2, "NOsh_setupCalcMGAUTO: min coarse = %g %g %g\n",
01755                     minc[0], minc[1], minc[2]);
01756             VASSERT(0);
01757         } else {
01758             Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): ifocus = %d, \
01759 fixing mesh min violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
01760             d[j], j);
01761             calcf->mgparm->center[j] += d[j];
01762             dofix = 1;
01763         }
01764     }
01765     /* Check if we've fallen off of the upper end of the mesh */
01766     maxf[j] = calcf->mgparm->center[j] \
01767         + 0.5*(calcf->mgparm->glen[j]);
01768     maxc[j] = calcc->mgparm->center[j] \
01769         + 0.5*(calcc->mgparm->glen[j]);
01770     d[j] = maxf[j] - maxc[j];
01771     if (d[j] >= VSMALL) {
01772         if (ifocus == (nfocus-1)) {
01773             Vnm_print(2, "NOsh_setupCalcMGAUTO: Error! Finest \
01774 mesh has fallen off the coarser meshes!\n");
01775             Vnm_print(2, "NOsh_setupCalcMGAUTO: difference in %d-\
01776 direction = %g\n", j, d[j]);
01777             VASSERT(0);
01778         } else {
01779             /* If we already fixed the lower boundary and we now need
01780             to fix the upper boundary, we have a serious problem. */
01781             if (dofix) {
01782                 Vnm_print(2, "NOsh_setupCalcMGAUTO: Error! Both \
01783 ends of the finer mesh do not fit in the bigger mesh!\n");
01784                 VASSERT(0);
01785             }
01786             Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d): ifocus = %d, \
01787 fixing mesh max violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
01788             d[j], j);
01789             calcf->mgparm->center[j] -= d[j];
01790             dofix = 1;
01791         }
01792     }
01793 }
01794 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): final fine mesh upper corner = \
01795 %g %g %g\n", __FILE__, __LINE__,
01796             calcf->mgparm->center[0]+0.5*(calcf->mgparm->glen[0]),
01797             calcf->mgparm->center[1]+0.5*(calcf->mgparm->glen[1]),
01798             calcf->mgparm->center[2]+0.5*(calcf->mgparm->glen[2]));
01799 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): final fine mesh lower corner = \
01800 %g %g %g\n", __FILE__, __LINE__,

```

```

01801     calcf->mgparm->center[0]-0.5*(calcf->mgparm->glen[0]),
01802     calcf->mgparm->center[1]-0.5*(calcf->mgparm->glen[1]),
01803     calcf->mgparm->center[2]-0.5*(calcf->mgparm->glen[2]));
01804 }
01805
01806 /* Finer levels have focusing boundary conditions */
01807 if (ifocus != 0) calcf->pbeparm->bclf = BCFL_FOCUS;
01808
01809 /* Only the finest level handles I/O and needs to worry about disjoint
01810 partitioning */
01811 if (ifocus != (nfocus-1)) calcf->pbeparm->numwrite = 0;
01812
01813 /* Reset boundary flags for everything except parallel focusing */
01814 if (calcf->mgparm->type != MCT_PARALLEL) {
01815 Vnm_print(0, "NOSH_SetupMGAUTO: Resetting boundary flags\n");
01816 for (j=0; j<6; j++) calcf->mgparm->partDisjOwnSide[j] = 0;
01817 for (j=0; j<3; j++) {
01818     calcf->mgparm->partDisjCenter[j] = 0;
01819     calcf->mgparm->partDisjLength[j] = calcf->mgparm->glen[j];
01820 }
01821 }
01822
01823
01824     calcf->mgparm->parsed = 1;
01825 }
01826
01827
01828     return 1;
01829 }
01830
01831 /* Author: Nathan Baker and Todd Dolinsky */
01832 VPUBLIC int NOSH_SetupCalcMGPARA(
01833     NOSh *thee,
01834     NOSh_calc *elec
01835 ) {
01836
01837 /* NEW (25-Jul-2006): This code should produce modify the ELEC statement
01838 and pass it on to MGAUTO for further processing. */
01839
01840 MGparm *mgparm = VNULL;
01841 double ofrac;
01842 double hx, hy, hzed;
01843 double xofrac, yofrac, zofrac;
01844 int rank, size, npx, npy, npz, nproc, ip, jp, kp;
01845 int xeffGlob, yeffGlob, zeffGlob, xDisj, yDisj, zDisj;
01846 int xigminDisj, xigmaxDisj, yigminDisj, yigmaxDisj, zigminDisj, zigmaxDisj;
01847 int xigminOlap, xigmaxOlap, yigminOlap, yigmaxOlap, zigminOlap, zigmaxOlap;
01848 int xOlapReg, yOlapReg, zOlapReg;
01849 double xlenDisj, ylenDisj, zlenDisj;
01850 double xcentDisj, ycentDisj, zcentDisj;
01851 double xcentOlap, ycentOlap, zcentOlap;
01852 double xlenOlap, ylenOlap, zlenOlap;
01853 double xminOlap, xmaxOlap, yminOlap, ymaxOlap, zminOlap, zmaxOlap;
01854 double xminDisj, xmaxDisj, yminDisj, ymaxDisj, zminDisj, zmaxDisj;
01855 double xcent, ycent, zcent;
01856
01857 /* Grab some useful variables */

```

```

01858     VASSERT(thee != VNULL);
01859     VASSERT(elec != VNULL);
01860     mgparm = elec->mgparm;
01861     VASSERT(mgparm != VNULL);
01862
01863     /* Grab some useful variables */
01864     ofrac = mgparm->ofrac;
01865     npx = mgparm->pdime[0];
01866     npy = mgparm->pdime[1];
01867     npz = mgparm->pdime[2];
01868     nproc = npx*npy*npz;
01869
01870     /* If this is not an asynchronous calculation, then we need to make sure we
01871        have all the necessary MPI information */
01872     if (mgparm->setasync == 0) {
01873
01874 #ifndef HAVE_MPI_H
01875
01876         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Oops!  You're trying to perform \
01877 an 'mg-para' (parallel) calculation\n");
01878         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  with a version of APBS that wasn't \
01879 compiled with MPI!\n");
01880         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Perhaps you meant to use the \
01881 'async' flag?\n");
01882         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Bailing out!\n");
01883
01884         return 0;
01885
01886 #endif
01887
01888     rank = thee->proc_rank;
01889     size = thee->proc_size;
01890     Vnm_print(0, "NOsh_setupCalcMGPARA:  Hello from processor %d of %d\n",
01891                 rank,
01892                 size);
01893
01894     /* Check to see if we have too many processors.  If so, then simply set
01895        this processor to duplicating the work of processor 0. */
01896     if (rank > (nproc-1)) {
01897         Vnm_print(2, "NOsh_setupMGPARA:  There are more processors available than\
01898 the %d you requested.\n", nproc);
01899         Vnm_print(2, "NOsh_setupMGPARA:  Eliminating processor %d\n", rank);
01900         thee->bogus = 1;
01901         rank = 0;
01902     }
01903
01904     /* Check to see if we have too few processors.  If so, this is a fatal
01905        error. */
01906     if (size < nproc) {
01907         Vnm_print(2, "NOsh_setupMGPARA:  There are too few processors (%d) to \
01908 satisfy requirements (%d)\n", size, nproc);
01909         return 0;
01910     }
01911     Vnm_print(0, "NOsh_setupMGPARA:  Hello (again) from processor %d of %d\n",
01912                 rank, size);
01913

```

```

01914     } else { /* Setting up for an asynchronous calculation. */
01915
01916     rank = mgparm->async;
01917
01918     thee->ispara = 1;
01919     thee->proc_rank = rank;
01920
01921         /* Check to see if the async id is greater than the number of
01922 * processors. If so, this is a fatal error. */
01923         if (rank > (nproc-1)) {
01924             Vnm_print(2, "NOsh_setupMGPARA: The processor id you requested (%d)
01925 is not within the range of processors available (0-%d)\n", rank, (nproc-1));
01926             return 0;
01927         }
01928     }
01929
01930     /* Calculate the processor's coordinates in the processor grid */
01931     kp = (int)floor(rank/(np*x*npy));
01932     jp = (int)floor((rank-kp*np*x*npy)/np*x);
01933     ip = rank - kp*np*x*npy - jp*np*x;
01934     Vnm_print(0, "NOsh_setupMGPARA: Hello world from PE (%d, %d, %d)\n",
01935             ip, jp, kp);
01936
01937     /* Calculate effective overlap fractions for uneven processor distributions */
01938     if (np*x == 1) xofrac = 0.0;
01939     else xofrac = ofrac;
01940     if (npy == 1) yofrac = 0.0;
01941     else yofrac = ofrac;
01942     if (npz == 1) zofrac = 0.0;
01943     else zofrac = ofrac;
01944
01945     /* Calculate the global grid size and spacing */
01946     xDisj = (int)VLOOR(mgparm->dime[0]/(1 + 2*xofrac) + 0.5);
01947     xeffGlob = np*x*xDisj;
01948     hx = mgparm->fglen[0]/(double)(xeffGlob-1);
01949     yDisj = (int)VLOOR(mgparm->dime[1]/(1 + 2*yofrac) + 0.5);
01950     yeffGlob = np*y*yDisj;
01951     hy = mgparm->fglen[1]/(double)(yeffGlob-1);
01952     zDisj = (int)VLOOR(mgparm->dime[2]/(1 + 2*zofrac) + 0.5);
01953     zeffGlob = npz*zDisj;
01954     hzed = mgparm->fglen[2]/(double)(zeffGlob-1);
01955     Vnm_print(0, "NOsh_setupMGPARA: Global Grid size = (%d, %d, %d)\n",
01956             xeffGlob, yeffGlob, zeffGlob);
01957     Vnm_print(0, "NOsh_setupMGPARA: Global Grid Spacing = (%.3f, %.3f, %.3f)\n",
01958             hx, hy, hzed);
01959     Vnm_print(0, "NOsh_setupMGPARA: Processor Grid Size = (%d, %d, %d)\n",
01960             xDisj, yDisj, zDisj);
01961
01962     /* Calculate the maximum and minimum processor grid points */
01963     xigminDisj = ip*xDisj;
01964     xigmaxDisj = xigminDisj + xDisj - 1;
01965     yigminDisj = jp*yDisj;
01966     yigmaxDisj = yigminDisj + yDisj - 1;
01967     zigminDisj = kp*zDisj;
01968     zigmaxDisj = zigminDisj + zDisj - 1;

```

```

01969     Vnm_print(0, "NOsh_setupMGPARA: Min Grid Points for this proc. (%d, %d, %d) \
n",
01970                 xigminDisj, yigminDisj, zigminDisj);
01971     Vnm_print(0, "NOsh_setupMGPARA: Max Grid Points for this proc. (%d, %d, %d) \
n",
01972                 xigmaxDisj, yigmaxDisj, zigmaxDisj);
01973
01974
01975     /* Calculate the disjoint partition length and center displacement */
01976     xminDisj = VMAX2(hx*(xigminDisj-0.5), 0.0);
01977     xmaxDisj = VMIN2(hx*(xigmaxDisj+0.5), mgparm->fglen[0]);
01978     xlenDisj = xmaxDisj - xminDisj;
01979     yminDisj = VMAX2(hy*(yigminDisj-0.5), 0.0);
01980     ymaxDisj = VMIN2(hy*(yigmaxDisj+0.5), mgparm->fglen[1]);
01981     ylenDisj = ymaxDisj - yminDisj;
01982     zminDisj = VMAX2(hzed*(zigminDisj-0.5), 0.0);
01983     zmaxDisj = VMIN2(hzed*(zigmaxDisj+0.5), mgparm->fglen[2]);
01984     zlenDisj = zmaxDisj - zminDisj;
01985
01986     xcent = 0.5*mgparm->fglen[0];
01987     ycent = 0.5*mgparm->fglen[1];
01988     zcent = 0.5*mgparm->fglen[2];
01989
01990     xcentDisj = xminDisj + 0.5*xlenDisj - xcent;
01991     ycentDisj = yminDisj + 0.5*ylenDisj - ycent;
01992     zcentDisj = zminDisj + 0.5*zlenDisj - zcent;
01993     if (VABS(xcentDisj) < VSMALL) xcentDisj = 0.0;
01994     if (VABS(ycentDisj) < VSMALL) ycentDisj = 0.0;
01995     if (VABS(zcentDisj) < VSMALL) zcentDisj = 0.0;
01996
01997     Vnm_print(0, "NOsh_setupMGPARA: Disj part length = (%g, %g, %g)\n",
01998                 xlenDisj, ylenDisj, zlenDisj);
01999     Vnm_print(0, "NOsh_setupMGPARA: Disj part center displacement = (%g, %g, %g)
n",
02000                 xcentDisj, ycentDisj, zcentDisj);
02001
02002     /* Calculate the overlapping partition length and center displacement */
02003     xOlapReg = 0;
02004     yOlapReg = 0;
02005     zOlapReg = 0;
02006     if (npx != 1) xOlapReg = (int)VFLLOOR(xofrac*mgparm->fglen[0]/npx/hx + 0.5) +
1;
02007     if (npy != 1) yOlapReg = (int)VFLLOOR(yofrac*mgparm->fglen[1]/npy/hy + 0.5) +
1;
02008     if (npz != 1) zOlapReg = (int)VFLLOOR(zofrac*mgparm->fglen[2]/npz/hzed + 0.5) +
1;
02009
02010     Vnm_print(0, "NOsh_setupMGPARA: No. of Grid Points in Overlap (%d, %d, %d)\n",
02011                 xOlapReg, yOlapReg, zOlapReg);
02012
02013     if (ip == 0) xigminOlap = 0;
02014     else if (ip == (npx - 1)) xigminOlap = xeffGlob - mgparm->dime[0];
02015     else xigminOlap = xigminDisj - xOlapReg;
02016     xigmaxOlap = xigminOlap + mgparm->dime[0] - 1;
02017
02018     if (jp == 0) yigminOlap = 0;

```

```

02019     else if (jp == (npy - 1)) yigminOlap = yeffGlob - mgparm->dime[1];
02020     else yigminOlap = yigminDisj - yOlapReg;
02021     yigmaxOlap = yigminOlap + mgparm->dime[1] - 1;
02022
02023     if (kp == 0) zigminOlap = 0;
02024     else if (kp == (npz - 1)) zigminOlap = zeffGlob - mgparm->dime[2];
02025     else zigminOlap = zigminDisj - zOlapReg;
02026     zigmaxOlap = zigminOlap + mgparm->dime[2] - 1;
02027
02028     Vnm_print(0, "NOsh_setupMGPARA: Min Grid Points with Overlap (%d, %d, %d)\n"
02029
02030             xigminOlap, yigminOlap, zigminOlap);
02031     Vnm_print(0, "NOsh_setupMGPARA: Max Grid Points with Overlap (%d, %d, %d)\n"
02032
02033             xigmaxOlap, yigmaxOlap, zigmaxOlap);
02034
02035     xminOlap = hx * xigminOlap;
02036     xmaxOlap = hx * xigmaxOlap;
02037     yminOlap = hy * yigminOlap;
02038     ymaxOlap = hy * yigmaxOlap;
02039     zminOlap = hzed * zigminOlap;
02040     zmaxOlap = hzed * zigmaxOlap;
02041
02042     xlenOlap = xmaxOlap - xminOlap;
02043     ylenOlap = ymaxOlap - yminOlap;
02044     zlenOlap = zmaxOlap - zminOlap;
02045
02046     xcentOlap = (xminOlap + 0.5*xlenOlap) - xcent;
02047     ycentOlap = (yminOlap + 0.5*ylenOlap) - ycent;
02048     zcentOlap = (zminOlap + 0.5*zlenOlap) - zcent;
02049     if (VABS(xcentOlap) < VSMALL) xcentOlap = 0.0;
02050     if (VABS(ycentOlap) < VSMALL) ycentOlap = 0.0;
02051     if (VABS(zcentOlap) < VSMALL) zcentOlap = 0.0;
02052
02053     Vnm_print(0, "NOsh_setupMGPARA: Olap part length = (%g, %g, %g)\n",
02054             xlenOlap, ylenOlap, zlenOlap);
02055     Vnm_print(0, "NOsh_setupMGPARA: Olap part center displacement = (%g, %g, %g)
02056
02057             \n",
02058             xcentOlap, ycentOlap, zcentOlap);
02059
02060
02061     /* Calculate the boundary flags:
02062     Flags are set to 1 when another processor is present along the boundary
02063     Flags are otherwise set to 0. */
02064
02065     if (ip == 0) mgparm->partDisjOwnSide[VAPBS_LEFT] = 0;
02066     else mgparm->partDisjOwnSide[VAPBS_LEFT] = 1;
02067     if (ip == (npx-1)) mgparm->partDisjOwnSide[VAPBS_RIGHT] = 0;
02068     else mgparm->partDisjOwnSide[VAPBS_RIGHT] = 1;
02069     if (jp == 0) mgparm->partDisjOwnSide[VAPBS_BACK] = 0;
02070     else mgparm->partDisjOwnSide[VAPBS_BACK] = 1;
02071     if (jp == (npz-1)) mgparm->partDisjOwnSide[VAPBS_FRONT] = 0;
02072     else mgparm->partDisjOwnSide[VAPBS_FRONT] = 1;
02073     if (kp == 0) mgparm->partDisjOwnSide[VAPBS_DOWN] = 0;
02074     else mgparm->partDisjOwnSide[VAPBS_DOWN] = 1;
02075     if (kp == (npz-1)) mgparm->partDisjOwnSide[VAPBS_UP] = 0;
02076     else mgparm->partDisjOwnSide[VAPBS_UP] = 1;

```

```

02073
02074 Vnm_print(0, "NOsh_setupMGPARA: partDisjOwnSide[LEFT] = %d\n",
02075     mgparm->partDisjOwnSide[VAPBS_LEFT]);
02076 Vnm_print(0, "NOsh_setupMGPARA: partDisjOwnSide[RIGHT] = %d\n",
02077     mgparm->partDisjOwnSide[VAPBS_RIGHT]);
02078 Vnm_print(0, "NOsh_setupMGPARA: partDisjOwnSide[FRONT] = %d\n",
02079     mgparm->partDisjOwnSide[VAPBS_FRONT]);
02080 Vnm_print(0, "NOsh_setupMGPARA: partDisjOwnSide[BACK] = %d\n",
02081     mgparm->partDisjOwnSide[VAPBS_BACK]);
02082 Vnm_print(0, "NOsh_setupMGPARA: partDisjOwnSide[UP] = %d\n",
02083     mgparm->partDisjOwnSide[VAPBS_UP]);
02084 Vnm_print(0, "NOsh_setupMGPARA: partDisjOwnSide[DOWN] = %d\n",
02085     mgparm->partDisjOwnSide[VAPBS_DOWN]);
02086
02087 /* Set the mesh parameters */
02088 mgparm->frlen[0] = xlenOlap;
02089 mgparm->frlen[1] = ylenOlap;
02090 mgparm->frlen[2] = zlenOlap;
02091 mgparm->partDisjLength[0] = xlenDisj;
02092 mgparm->partDisjLength[1] = ylenDisj;
02093 mgparm->partDisjLength[2] = zlenDisj;
02094 mgparm->partDisjCenter[0] = mgparm->fcenter[0] + xcentDisj;
02095 mgparm->partDisjCenter[1] = mgparm->fcenter[1] + ycentDisj;
02096 mgparm->partDisjCenter[2] = mgparm->fcenter[2] + zcentDisj;
02097 mgparm->fcenter[0] += xcentOlap;
02098 mgparm->fcenter[1] += ycentOlap;
02099 mgparm->fcenter[2] += zcentOlap;
02100
02101 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d): Set up *relative* partition \
02102 centers...\n", __FILE__, __LINE__);
02103 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d): Absolute centers will be set \
02104 in NOsh_setupMGAUTO\n", __FILE__, __LINE__);
02105 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d): partDisjCenter = %g %g %g\n",
02106     __FILE__, __LINE__,
02107     mgparm->partDisjCenter[0],
02108     mgparm->partDisjCenter[1],
02109     mgparm->partDisjCenter[2]);
02110 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d): ccenter = %g %g %g\n",
02111     __FILE__, __LINE__,
02112     mgparm->ccenter[0],
02113     mgparm->ccenter[1],
02114     mgparm->ccenter[2]);
02115 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d): fcenter = %g %g %g\n",
02116     __FILE__, __LINE__,
02117     mgparm->fcenter[0],
02118     mgparm->fcenter[1],
02119     mgparm->fcenter[2]);
02120
02121
02122 /* Setup the automatic focusing calculations associated with this processor */
02123 return NOsh_setupCalcMGAUTO(thee, elec);
02124
02125 }
02126
02127 VPUBLIC int NOsh_parseFEM(
02128     NOsh *thee,
02129     Vio *sock,

```

```

02130           NOsh_calc *elec
02131       ) {
02132
02133   char tok[VMAX_BUFSIZE];
02134   FEMparm *feparm = VNULL;
02135   PBEparm *pbeparm = VNULL;
02136   int rc;
02137   Vrc_Codes vrc;
02138
02139 /* Check the arguments */
02140 if (thee == VNULL) {
02141   Vnm_print(2, "NOsh_parseFEM: Got NULL thee!\n");
02142   return 0;
02143 }
02144 if (sock == VNULL) {
02145   Vnm_print(2, "NOsh_parseFEM: Got pointer to NULL socket!\n");
02146   return 0;
02147 }
02148 if (elec == VNULL) {
02149   Vnm_print(2, "NOsh_parseFEM: Got pointer to NULL elec object!\n");
02150   return 0;
02151 }
02152 feparm = elec->feparm;
02153 if (feparm == VNULL) {
02154   Vnm_print(2, "NOsh_parseFEM: Got pointer to NULL feparm object!\n");
02155   return 0;
02156 }
02157 pbeparm = elec->pbeparm;
02158 if (pbeparm == VNULL) {
02159   Vnm_print(2, "NOsh_parseFEM: Got pointer to NULL pbeparm object!\n");
02160   return 0;
02161 }
02162
02163 Vnm_print(0, "NOsh_parseFEM: Parsing parameters for FEM calculation\n");
02164
02165 /* Start snarfing tokens from the input stream */
02166 rc = 1;
02167 while (Vio_scanf(sock, "%s", tok) == 1) {
02168
02169   Vnm_print(0, "NOsh_parseFEM: Parsing %s...\n", tok);
02170
02171 /* See if it's an END token */
02172 if (Vstring_strcasecmp(tok, "end") == 0) {
02173   feparm->parsed = 1;
02174   pbeparm->parsed = 1;
02175   rc = 1;
02176   break;
02177 }
02178
02179 /* Pass the token through a series of parsers */
02180 rc = PBEparm_parseToken(pbeparm, tok, sock);
02181 if (rc == -1) {
02182   Vnm_print(0, "NOsh_parseFEM: parsePBE error!\n");
02183   break;
02184 } else if (rc == 0) {
02185   /* Pass the token to the generic MG parser */
02186   vrc = FEMparm_parseToken(feparm, tok, sock);

```

```

02187     if (vrc == VRC_FAILURE) {
02188         Vnm_print(0, "NOsh_parseFEM:  parseMG error!\n");
02189         break;
02190     } else if (vrc == VRC_WARNING) {
02191         /* We ran out of parsers! */
02192         Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
02193         break;
02194     }
02195 }
02196 }
02197
02198 /* Handle various errors arising in the token-snarfing loop -- these all
02199 * just result in simple returns right now */
02200 if (rc == -1) return 0;
02201 if (rc == 0) return 0;
02202
02203 /* Check the status of the parameter objects */
02204 if ((!FEMparm_check(feparm)) || (!PBEparm_check(pbeparm))) {
02205     Vnm_print(2, "NOsh:  FEM parameters not set correctly!\n");
02206     return 0;
02207 }
02208
02209 return 1;
02210
02211 }
02212
02213 VPRIPRIVATE int NOsh_setupCalcFEMANUAL(
02214     NOsh *thee,
02215     NOsh_calc *elec
02216     ) {
02217
02218     FEMparm *feparm = VNULL;
02219     PBEparm *pbeparm = VNULL;
02220     NOsh_calc *calc = VNULL;
02221
02222     VASSERT(thee != VNULL);
02223     VASSERT(elec != VNULL);
02224     feparm = elec->feparm;
02225     VASSERT(feparm != VNULL);
02226     pbeparm = elec->pbeparm;
02227     VASSERT(pbeparm);
02228
02229     /* Check to see if he have any room left for this type of
02230      * calculation, if so: set the calculation type, update the number
02231      * of calculations of this type, and parse the rest of the section
02232      */
02233     if (thee->nCalc >= NOSH_MAXCALC) {
02234         Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02235         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02236             NOSH_MAXCALC);
02237         return 0;
02238     }
02239     thee->calc[thee->nCalc] = NOsh_calc_ctor(NCT_FEM);
02240     calc = thee->calc[thee->nCalc];
02241     (thee->nCalc)++;
02242
02243     /* Copy over contents of ELEC */

```

```

02244 NOsh_calc_copy(calc, elec);
02245
02246
02247 return 1;
02248 }
02249
02250 VPUBLIC int NOsh_parseAPOL(
02251     NOsh *thee,
02252     Vio *sock,
02253     NOsh_calc *elec
02254     ) {
02255
02256     char tok[VMAX_BUFSIZE];
02257     APOLparm *apolparm = VNULL;
02258     int rc;
02259
02260     /* Check the arguments */
02261     if (thee == VNULL) {
02262         Vnm_print(2, "NOsh_parseAPOL: Got NULL thee!\n");
02263         return 0;
02264     }
02265     if (sock == VNULL) {
02266         Vnm_print(2, "NOsh_parseAPOL: Got pointer to NULL socket!\n");
02267         return 0;
02268     }
02269     if (elec == VNULL) {
02270         Vnm_print(2, "NOsh_parseAPOL: Got pointer to NULL elec object!\n");
02271         return 0;
02272     }
02273     apolparm = elec->apolparm;
02274     if (apolparm == VNULL) {
02275         Vnm_print(2, "NOsh_parseAPOL: Got pointer to NULL feparm object!\n");
02276         return 0;
02277     }
02278
02279     Vnm_print(0, "NOsh_parseAPOL: Parsing parameters for APOL calculation\n");
02280
02281     /* Start snarfing tokens from the input stream */
02282     rc = 1;
02283     while (Vio_scanf(sock, "%s", tok) == 1) {
02284
02285         Vnm_print(0, "NOsh_parseAPOL: Parsing %s...\n", tok);
02286         /* See if it's an END token */
02287         if (Vstring_strcasecmp(tok, "end") == 0) {
02288             apolparm->parsed = 1;
02289             rc = 1;
02290             break;
02291         }
02292
02293         /* Pass the token through a series of parsers */
02294         /* Pass the token to the generic non-polar parser */
02295         rc = APOLparm_parseToken(apolparm, tok, sock);
02296         if (rc == -1) {
02297             Vnm_print(0, "NOsh_parseFEM: parseMG error!\n");
02298             break;
02299         } else if (rc == 0) {
02300             /* We ran out of parsers! */

```

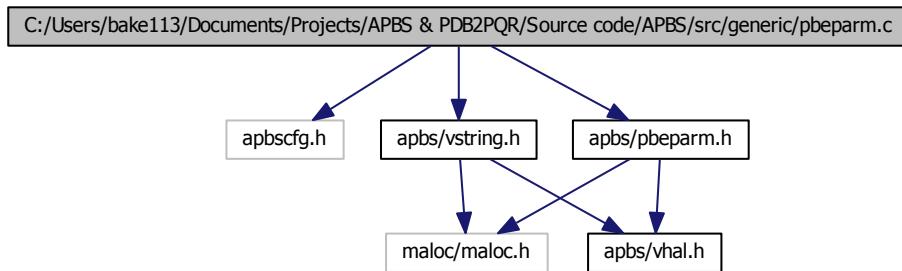
```
02301     Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
02302     break;
02303 }
02304
02305 }
02306
02307 /* Handle various errors arising in the token-snarfing loop -- these all
02308 * just result in simple returns right now */
02309 if (rc == -1) return 0;
02310 if (rc == 0) return 0;
02311
02312 /* Check the status of the parameter objects */
02313 if (!APOLparm_check(apolparm)) {
02314     Vnm_print(2, "NOsh:  APOL parameters not set correctly!\n");
02315     return 0;
02316 }
02317
02318 return 1;
02319
02320 }
02321
02322 VPRIVATE int NOsh_setupCalcAPOL(
02323     NOsh *thee,
02324     NOsh_calc *apol
02325 ) {
02326
02327     NOsh_calc *calc = VNULL;
02328
02329     VASSERT(thee != VNULL);
02330     VASSERT(apol != VNULL);
02331
02332     /* Check to see if he have any room left for this type of
02333      * calculation, if so: set the calculation type, update the number
02334      * of calculations of this type, and parse the rest of the section
02335      */
02336     if (thee->nCalc >= NOSH_MAXCALC) {
02337         Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02338         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02339             NOSH_MAXCALC);
02340         return 0;
02341     }
02342     thee->calc[thee->nCalc] = NOsh_calc_ctor(NCT_APOL);
02343     calc = thee->calc[thee->nCalc];
02344     (thee->nCalc)++;
02345
02346     /* Copy over contents of APOL */
02347     NOsh_calc_copy(calc, apol);
02348
02349     return 1;
02350 }
```

10.59 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/pbeparm.c File Reference

Class PBParm methods.

```
#include "apbscfg.h"
#include "apbs/pbeparm.h"
#include "apbs/vstring.h"

Include dependency graph for pbeparm.c:
```



Functions

- VPUBLIC double [PBParm_getIonCharge](#) (PBParm *thee, int i)
Get charge (e) of specified ion species.
- VPUBLIC double [PBParm_getIonConc](#) (PBParm *thee, int i)
Get concentration (M) of specified ion species.
- VPUBLIC double [PBParm_getIonRadius](#) (PBParm *thee, int i)
Get radius (A) of specified ion species.
- VPUBLIC double [PBParm_getZmem](#) (PBParm *thee)
- VPUBLIC double [PBParm_getLmem](#) (PBParm *thee)
- VPUBLIC double [PBParm_getMembraneDielectric](#) (PBParm *thee)
- VPUBLIC double [PBParm_getMemv](#) (PBParm *thee)
- VPUBLIC PBParm * [PBParm_ctor](#) ()
Construct PBParm object.
- VPUBLIC int [PBParm_ctor2](#) (PBParm *thee)
FORTRAN stub to construct PBParm object.

- VPUBLIC void **PBEPARM_DTOR** (**PBEPARM** **thee)
Object destructor.
- VPUBLIC void **PBEPARM_DTOR2** (**PBEPARM** *thee)
FORTRAN stub for object destructor.
- VPUBLIC int **PBEPARM_CHECK** (**PBEPARM** *thee)
Consistency check for parameter values stored in object.
- VPUBLIC void **PBEPARM_COPY** (**PBEPARM** *thee, **PBEPARM** *parm)
Copy PBEPARM object into thee.
 - VPRIVATE int **PBEPARM_PARSELPBE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSENPBE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSEMOL** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSELRPBE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSENRPBE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSESMPBE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSEBCFL** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSESELON** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSEPDIE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSESDENS** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSESDIE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSESRFM** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSESRAD** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSESWIN** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSETEMP** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSEUSEMAP** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSECALCENERGY** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSECALCFORCE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSEZMEM** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSELMEM** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSEMDIE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSEMEMV** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSEWRITE** (**PBEPARM** *thee, **VIO** *sock)
 - VPRIVATE int **PBEPARM_PARSEWRITEMAT** (**PBEPARM** *thee, **VIO** *sock)
 - VPUBLIC int **PBEPARM_PARSETOKEN** (**PBEPARM** *thee, **char** tok[VMAX_BUFSIZE],
VIO *sock)

Parse a keyword from an input file.

10.59.1 Detailed Description

Class PBEParm methods.

Author

Nathan Baker

Version

Id:

[pbeparm.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [pbeparm.c](#).

10.60 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/pbeparm.c

```
00001
00057 #include "apbscfg.h"
00058 #include "apbs/pbeparm.h"
00059 #include "apbs/vstring.h"
00060
00061 VEMBED(rcsid="$Id: pbeparm.c 1667 2011-12-02 23:22:02Z pcellis $")
00062
00063 #if !defined(VINLINE_MGPARM)
00064
00065 #endif /* if !defined(VINLINE_MGPARM) */
00066
00067 VPUBLIC double PBParm_getIonCharge(PBParm *thee, int i) {
00068     VASSERT(thee != VNULL);
00069     VASSERT(i < thee->nion);
00070     return thee->ionq[i];
00071 }
00072
00073 VPUBLIC double PBParm_getIonConc(PBParm *thee, int i) {
00074     VASSERT(thee != VNULL);
00075     VASSERT(i < thee->nion);
00076     return thee->ionc[i];
00077 }
00078
00079 VPUBLIC double PBParm_getIonRadius(PBParm *thee, int i) {
00080     VASSERT(thee != VNULL);
00081     VASSERT(i < thee->nion);
00082     return thee->ionr[i];
00083 }
00084
00085 /***** Added by Michael Grabe ****/
00086 /* Added by Michael Grabe */
00087 /***** */
00088
00089 VPUBLIC double PBParm_getzmem(PBParm *thee) {
00090     VASSERT(thee != VNULL);
00091     return thee->zmem;
00092 }
00093 VPUBLIC double PBParm_getLmem(PBParm *thee) {
00094     VASSERT(thee != VNULL);
```

```

00095     return thee->Lmem;
00096 }
00097 VPUBLIC double PBEparm_getmembraneDiel(PBEparm *thee) {
00098     VASSERT(thee != VNULL);
00099     return thee->mdie;
00100 }
00101 VPUBLIC double PBEparm_getmemv(PBEparm *thee) {
00102     VASSERT(thee != VNULL);
00103     return thee->memv;
00104 }
00105
00106 VPUBLIC PBEparm* PBEparm_ctor() {
00107
00108     /* Set up the structure */
00109     PBEparm *thee = VNULL;
00110     thee = Vmem_malloc(VNULL, 1, sizeof(PBEparm));
00111     VASSERT( thee != VNULL );
00112     VASSERT( PBEparm_ctor2(thee) );
00113
00114     return thee;
00115 }
00116
00117 VPUBLIC int PBEparm_ctor2(PBEparm *thee) {
00118
00119     int i;
00120
00121     if (thee == VNULL) return 0;
00122
00123     thee->parsed = 0;
00124
00125     thee->setmolid = 0;
00126     thee->setpbtype = 0;
00127     thee->setbcfl = 0;
00128     thee->setnion = 0;
00129     for (i=0; i<MAXION; i++) {
00130         thee->setion[i] = 0;
00131         thee->ionq[i] = 0.0;
00132         thee->ionc[i] = 0.0;
00133         thee->ionr[i] = 0.0;
00134     }
00135     thee->setpdie = 0;
00136     thee->setsdie = 0;
00137     thee->setsrfm = 0;
00138     thee->setsrad = 0;
00139     thee->setswin = 0;
00140     thee->settemp = 0;
00141     thee->setcalcenergy = 0;
00142     thee->setcalcforce = 0;
00143     thee->setsdens = 0;
00144     thee->numwrite = 0;
00145     thee->setwritemat = 0;
00146     thee->nion = 0;
00147     thee->sdens = 0;
00148     thee->swin = 0;
00149     thee->srad = 1.4;
00150     thee->useDielMap = 0;
00151     thee->useKappaMap = 0;

```

```

00152     thee->usePotMap = 0;
00153     thee->useChargeMap = 0;
00154
00155     /*-----*/
00156     /* Added by Michael Grabe */
00157     /*-----*/
00158
00159     thee->setZmem = 0;
00160     thee->setLmem = 0;
00161     thee->setMDie = 0;
00162     thee->setMemv = 0;
00163
00164     /*-----*/
00165
00166     thee->smsize = 0.0;
00167     thee->smvolume = 0.0;
00168
00169     thee->setSMSIZE = 0;
00170     thee->setSMVOLUME = 0;
00171
00172     return 1;
00173 }
00174
00175 VPUBLIC void PBEPARM_dtor(PBEPARM **thee) {
00176     if ((*thee) != VNULL) {
00177         PBEPARM_dtor2(*thee);
00178         Vmem_free(VNULL, 1, sizeof(PBEPARM), (void **)thee);
00179         (*thee) = VNULL;
00180     }
00181 }
00182
00183 VPUBLIC void PBEPARM_dtor2(PBEPARM *thee) { ; }
00184
00185 VPUBLIC int PBEPARM_check(PBEPARM *thee) {
00186
00187     int i;
00188
00189     /* Check to see if we were even filled... */
00190     if (!thee->parsed) {
00191         Vnm_print(2, "PBEPARM_check: not filled!\n");
00192         return 0;
00193     }
00194
00195     if (!thee->setMolid) {
00196         Vnm_print(2, "PBEPARM_check: MOL not set!\n");
00197         return 0;
00198     }
00199     if (!thee->setPBETYPE) {
00200         Vnm_print(2, "PBEPARM_check: LPBE/NPBE/LRPBE/NRPBE/SMPBE not set!\n");
00201         return 0;
00202     }
00203     if (!thee->setBCFL) {
00204         Vnm_print(2, "PBEPARM_check: BCFL not set!\n");
00205         return 0;
00206     }
00207     if (!thee->setNION) {
00208         thee->setNION = 1;

```

```

00209     thee->nion = 0;
00210 }
00211 for (i=0; i<thee->nion; i++) {
00212     if (!thee->setion[i]) {
00213         Vnm_print(2, "PBEparm_check: ION #%d not set!\n", i);
00214         return 0;
00215     }
00216 }
00217 if (!thee->setpdie) {
00218     Vnm_print(2, "PBEparm_check: PDIE not set!\n");
00219     return 0;
00220 }
00221 if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH)) \
00222     && (!thee->setsdens) && (thee->srad > VSMALL)) {
00223     Vnm_print(2, "PBEparm_check: SDENS not set!\n");
00224     return 0;
00225 }
00226 if (!thee->setsdie) {
00227     Vnm_print(2, "PBEparm_check: SDIE not set!\n");
00228     return 0;
00229 }
00230 if (!thee->setsrfm) {
00231     Vnm_print(2, "PBEparm_check: SRFM not set!\n");
00232     return 0;
00233 }
00234 if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH)) \
00235     && (!thee->setsrad)) {
00236     Vnm_print(2, "PBEparm_check: SRAD not set!\n");
00237     return 0;
00238 }
00239 if ((thee->srfm==VSM_SPLINE) && (!thee->setswin)) {
00240     Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00241     return 0;
00242 }
00243 if ((thee->srfm==VSM_SPLINE3) && (!thee->setswin)) {
00244     Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00245     return 0;
00246 }
00247 if ((thee->srfm==VSM_SPLINE4) && (!thee->setswin)) {
00248     Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00249     return 0;
00250 }
00251 if (!thee->settemp) {
00252     Vnm_print(2, "PBEparm_check: TEMP not set!\n");
00253     return 0;
00254 }
00255 if (!thee->setcalcenergy) thee->calcenergy = PCE_NO;
00256 if (!thee->setcalcforce) thee->calcforce = PCF_NO;
00257 if (!thee->setwritemat) thee->writemat = 0;
00258 /*-----*/
00259 /* Added by Michael Grabe */
00260 /*-----*/
00261 /*-----*/
00262 if ((!thee->setzmem) && (thee->bcfl == 3)){
00263     Vnm_print(2, "PBEparm_check: ZMEM not set!\n");
00264     return 0;
00265 
```

```

00266     }
00267     if ((!thee->setLmem) && (thee->bcfl == 3)){
00268         Vnm_print(2, "PBEparm_check: LMEM not set!\n");
00269         return 0;
00270     }
00271     if ((!thee->setmdie) && (thee->bcfl == 3)){
00272         Vnm_print(2, "PBEparm_check: MDIE not set!\n");
00273         return 0;
00274     }
00275     if ((!thee->setmemv) && (thee->bcfl == 3)){
00276         Vnm_print(2, "PBEparm_check: MEMV not set!\n");
00277         return 0;
00278     }
00279
00280 /*-----*/
00281     return 1;
00282 }
00283 }

00284 VPUBLIC void PBEparm_copy(PBEparm *thee, PBEparm *parm) {
00285
00286     int i, j;
00287
00288     VASSERT(thee != VNULL);
00289     VASSERT(parm != VNULL);
00290
00291     thee->molid = parm->molid;
00292     thee->setmolid = parm->setmolid;
00293     thee->useDielMap = parm->useDielMap;
00294     thee->dielMapID = parm->dielMapID;
00295     thee->useKappaMap = parm->useKappaMap;
00296     thee->kappaMapID = parm->kappaMapID;
00297
00298     thee->usePotMap = parm->usePotMap;
00299     thee->potMapID = parm->potMapID;
00300     thee->useChargeMap = parm->useChargeMap;
00301     thee->chargeMapID = parm->chargeMapID;
00302     thee->pbetype = parm->pbetype;
00303     thee->setpbetype = parm->setpbetype;
00304     thee->bcfl = parm->bcfl;
00305     thee->setbcfl = parm->setbcfl;
00306     thee->nion = parm->nion;
00307     thee->setnion = parm->setnion;
00308     for (i=0; i<MAXION; i++) {
00309         thee->ionq[i] = parm->ionq[i];
00310         thee->ionc[i] = parm->ionc[i];
00311         thee->ionr[i] = parm->ionr[i];
00312         thee->setion[i] = parm->setion[i];
00313     };
00314     thee->pdie = parm->pdie;
00315     thee->setpdie = parm->setpdie;
00316     thee->sdens = parm->sdens;
00317     thee->setsdens = parm->setsdens;
00318     thee->sdie = parm->sdie;
00319     thee->setsdie = parm->setsdie;
00320     thee->srfm = parm->srfm;
00321     thee->setsrfm = parm->setsrfm;
00322     thee->srad = parm->srad;

```

```

00323     thee->setsrad = parm->setsrad;
00324     thee->swin = parm->swin;
00325     thee->setswin = parm->setswin;
00326     thee->temp = parm->temp;
00327     thee->settemp = parm->settemp;
00328     thee->calcenergy = parm->calcenergy;
00329     thee->setcalcenergy = parm->setcalcenergy;
00330     thee->calcforce = parm->calcforce;
00331     thee->setcalcforce = parm->setcalcforce;
00332
00333     /*-----*/
00334     /* Added by Michael Grabe */
00335     /*-----*/
00336
00337     thee->zmem = parm->zmem;
00338     thee->setzmem = parm->setzmem;
00339     thee->Lmem = parm->Lmem;
00340     thee->setLmem = parm->setLmem;
00341     thee->mdie = parm->mdie;
00342     thee->setmdie = parm->setmdie;
00343     thee->memv = parm->memv;
00344     thee->setmemv = parm->setmemv;
00345
00346     /*-----*/
00347
00348     thee->numwrite = parm->numwrite;
00349     for (i=0; i<PBEPARM_MAXWRITE; i++) {
00350         thee->writetype[i] = parm->writetype[i];
00351         thee->writefmt[i] = parm->writefmt[i];
00352         for (j=0; j<VMAX_ARGLEN; j++)
00353             thee->writestem[i][j] = parm->writestem[i][j];
00354     }
00355     thee->writemat = parm->writemat;
00356     thee->setwritemat = parm->setwritemat;
00357     for (i=0; i<VMAX_ARGLEN; i++) thee->writematstem[i] = parm->writematstem[i];
00358     thee->writematflag = parm->writematflag;
00359
00360     thee->smsize = parm->smsize;
00361     thee->smvolume = parm->smvolume;
00362
00363     thee->setsmsize = parm->setsmsize;
00364     thee->setsmvolume = parm->setsmvolume;
00365
00366     thee->parsed = parm->parsed;
00367
00368 }
00369
00370 VPRIVATE int PBEparm_parseLPBE(PBEparm *thee, Vio *sock) {
00371     Vnm_print(0, "NOsh: parsed lpbe\n");
00372     thee->pbetype = PBE_LPBE;
00373     thee->setpbetype = 1;
00374     return 1;
00375 }
00376
00377 VPRIVATE int PBEparm_parseNPBE(PBEparm *thee, Vio *sock) {
00378     Vnm_print(0, "NOsh: parsed npbe\n");
00379     thee->pbetype = PBE_NPBE;

```

```

00380     thee->setpbetype = 1;
00381     return 1;
00382 }
00383
00384 VPRIVATE int PBparm_parseMOL(PBparm *thee, Vio *sock) {
00385     int ti;
00386     char tok[VMAX_BUFSIZE];
00387
00388     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00389     if (sscanf(tok, "%d", &ti) == 0) {
00390         Vnm_print(2, "NOsh: Read non-int (%s) while parsing MOL \
00391 keyword!\n", tok);
00392         return -1;
00393     }
00394     thee->molid = ti;
00395     thee->setmolid = 1;
00396     return 1;
00397
00398     VERROR1:
00399         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00400         return -1;
00401 }
00402
00403 VPRIVATE int PBparm_parseLRPBE(PBparm *thee, Vio *sock) {
00404     Vnm_print(0, "NOsh: parsed lrpbe\n");
00405     thee->pbetype = PBE_LRPBE;
00406     thee->setpbetype = 1;
00407     return 1;
00408 }
00409
00410 VPRIVATE int PBparm_parseNRPBE(PBparm *thee, Vio *sock) {
00411     Vnm_print(0, "NOsh: parsed nrpbe\n");
00412     thee->pbetype = PBE_NRPBE;
00413     thee->setpbetype = 1;
00414     return 1;
00415 }
00416
00417 VPRIVATE int PBparm_parseSMPBE(PBparm *thee, Vio *sock) {
00418
00419     int i;
00420
00421     char type[VMAX_BUFSIZE]; /* vol or size (keywords) */
00422     char value[VMAX_BUFSIZE]; /* floating point value */
00423
00424     char setVol = 1;
00425     char setSize = 1;
00426     char keyValuePairs = 2;
00427
00428     double size, volume;
00429
00430     for(i=0;i<keyValuePairs;i++) {
00431
00432         /* The line two tokens at a time */
00433         VJMPERR1(Vio_scanf(sock, "%s", type) == 1);
00434         VJMPERR1(Vio_scanf(sock, "%s", value) == 1);
00435
00436         if(!strcmp(type, "vol")) {

```

```

00437     if ((setVol = sscanf(value, "%lf", &volume)) == 0){
00438         Vnm_print(2,"NOsh: Read non-float (%s) while parsing smpbe keyword!\n", value
e);
00439         return VRC_FAILURE;
00440     }
00441     }else if(!strcmp(type,"size")){
00442     if ((setSize = sscanf(value, "%lf", &size)) == 0){
00443         Vnm_print(2,"NOsh: Read non-float (%s) while parsing smpbe keyword!\n", value
e);
00444         return VRC_FAILURE;
00445     }
00446     }else{
00447     Vnm_print(2,"NOsh: Read non-float (%s) while parsing smpbe keyword!\n", value
);
00448     return VRC_FAILURE;
00449 }
00450 }
00451
00452 /* If either the volume or size isn't set, throw an error */
00453 if((setVol == 0) || (setSize == 0)){
00454     Vnm_print(2,"NOsh: Error while parsing smpbe keywords! Only size or vol was sp
ecified.\n");
00455     return VRC_FAILURE;
00456 }
00457
00458 Vnm_print(0, "NOsh: parsed smpbe\n");
00459     thee->pbeptype = PBE_SMPBE;
00460     thee->setpbeptype = 1;
00461
00462     thee->smsize = size;
00463     thee->setsmsize = 1;
00464
00465     thee->smvolume = volume;
00466     thee->setsmvolume = 1;
00467
00468     return VRC_SUCCESS;
00469
00470 VERROR1:
00471     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00472     return VRC_FAILURE;
00473
00474 }
00475
00476 VPRIVATE int PBEparm_parseBCFL(PBEparm *thee, Vio *sock) {
00477     char tok[VMAX_BUFSIZE];
00478     int ti;
00479
00480     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00481
00482     /* We can either parse int flag... */
00483     if (sscanf(tok, "%d", &ti) == 1) {
00484
00485         thee->bcl = ti;
00486         thee->setbcl = 1;
00487         /* Warn that this usage is deprecated */
00488         Vnm_print(2, "parsePBE: Warning -- parsed deprecated \"bcl %d\" \
00489 statement\n", ti);

```

```

00490     Vnm_print(2, "parsePBE: Please use \"bcfl \"");
00491     switch (thee->bcfl) {
00492         case BCFL_ZERO:
00493             Vnm_print(2, "zero");
00494             break;
00495         case BCFL_SDH:
00496             Vnm_print(2, "sdh");
00497             break;
00498         case BCFL_MDH:
00499             Vnm_print(2, "mdh");
00500             break;
00501         case BCFL_FOCUS:
00502             Vnm_print(2, "focus");
00503             break;
00504         case BCFL_MEM:
00505             Vnm_print(2, "mem");
00506             break;
00507         case BCFL_MAP:
00508             Vnm_print(2, "map");
00509             break;
00510         default:
00511             Vnm_print(2, "UNKNOWN");
00512             break;
00513     }
00514     Vnm_print(2, "\" instead.\n");
00515     return 1;
00516
00517 /* ...or the word */
00518 } else {
00519
00520     if (Vstring_strcasecmp(tok, "zero") == 0) {
00521         thee->bcfl = BCFL_ZERO;
00522         thee->setbcfl = 1;
00523         return 1;
00524     } else if (Vstring_strcasecmp(tok, "sdh") == 0) {
00525         thee->bcfl = BCFL_SDH;
00526         thee->setbcfl = 1;
00527         return 1;
00528     } else if (Vstring_strcasecmp(tok, "mdh") == 0) {
00529         thee->bcfl = BCFL_MDH;
00530         thee->setbcfl = 1;
00531         return 1;
00532     } else if (Vstring_strcasecmp(tok, "focus") == 0) {
00533         thee->bcfl = BCFL_FOCUS;
00534         thee->setbcfl = 1;
00535         return 1;
00536     } else if (Vstring_strcasecmp(tok, "mem") == 0) {
00537         thee->bcfl = BCFL_MEM;
00538         thee->setbcfl = 1;
00539         return 1;
00540     } else if (Vstring_strcasecmp(tok, "map") == 0) {
00541         thee->bcfl = BCFL_MAP;
00542         thee->setbcfl = 1;
00543         return 1;
00544     } else {
00545         Vnm_print(2, "NOsh: parsed unknown BCFL parameter (%s) !\n",
00546         tok);

```

```

00547         return -1;
00548     }
00549 }
00550     return 0;
00551
00552     VERROR1:
00553     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00554     return -1;
00555 }
00556
00557 VPRIVATE int PBEparm_parseION(PBEparm *thee, Vio *sock) {
00558
00559     int i;
00560     int meth = 0;
00561
00562     char tok[VMAX_BUFSIZE];
00563     char value[VMAX_BUFSIZE];
00564
00565     double tf;
00566     double charge, conc, radius;
00567
00568     int setCharge = 0;
00569     int setConc = 0;
00570     int setRadius = 0;
00571     int keyValuePairs = 3;
00572
00573     /* Get the initial token for the ION statement */
00574     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00575
00576     /* Scan the token once to determine the type (old style or new key-value pair) */
00577
00578     meth = sscanf(tok, "%lf", &tf);
00579     /* If tok is a non-zero float value, we are using the old method */
00580     if(meth != 0){
00581
00582         Vnm_print(2, "NOsh: Deprecated use of ION keyword! Use key-value pairs\n", tok
00583     );
00584
00585         if (sscanf(tok, "%lf", &tf) == 0) {
00586             Vnm_print(2, "NOsh: Read non-float (%s) while parsing ION keyword!\n", tok);
00587             return VRC_FAILURE;
00588         }
00589         thee->ionq[thee->nion] = tf;
00590         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00591         if (sscanf(tok, "%lf", &tf) == 0) {
00592             Vnm_print(2, "NOsh: Read non-float (%s) while parsing ION keyword!\n", tok);
00593             return VRC_FAILURE;
00594         }
00595         thee->ionc[thee->nion] = tf;
00596         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00597         if (sscanf(tok, "%lf", &tf) == 0) {
00598             Vnm_print(2, "NOsh: Read non-float (%s) while parsing ION keyword!\n", tok);
00599             return VRC_FAILURE;
00600         }
00601     }else{

```

```

00602
00603 /* Three key-value pairs (charge, radius and conc) */
00604 for(i=0;i<keyValuePairs;i++){
00605
00606 /* Now scan for the value (float) to be used with the key token parsed
00607 * above the if-else statement */
00608 VJMPERR1(Vio_sccanf(sock, "%s", value) == 1);
00609 if(!strcmp(tok,"charge")){
00610     setCharge = sscanf(value, "%lf", &charge);
00611     if (setCharge == 0){
00612         Vnm_print(2,"NOsh: Read non-float (%s) while parsing ION %s keyword!\n", va
lue, tok);
00613         return VRC_FAILURE;
00614     }
00615     thee->ionq[thee->nion] = charge;
00616 }else if(!strcmp(tok,"radius")){
00617     setRadius = sscanf(value, "%lf", &radius);
00618     if (setRadius == 0){
00619         Vnm_print(2,"NOsh: Read non-float (%s) while parsing ION %s keyword!\n", va
lue, tok);
00620         return VRC_FAILURE;
00621     }
00622     thee->ionr[thee->nion] = radius;
00623 }else if(!strcmp(tok,"conc")){
00624     setConc = sscanf(value, "%lf", &conc);
00625     if (setConc == 0){
00626         Vnm_print(2,"NOsh: Read non-float (%s) while parsing ION %s keyword!\n", va
lue, tok);
00627         return VRC_FAILURE;
00628     }
00629     thee->ionc[thee->nion] = conc;
00630 }else{
00631     Vnm_print(2,"NOsh: Illegal or missing key-value pair for ION keyword!\n");
00632     return VRC_FAILURE;
00633 }
00634
00635 /* If all three values haven't be set (setValue = 0) then read the next token
*/
00636 if((setCharge != 1) || (setConc != 1) || (setRadius != 1)){
00637     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00638 }
00639
00640 } /* end for */
00641 } /* end if */
00642
00643 /* Finally set the setion, nion and setnion flags and return success */
00644 thee->setion[thee->nion] = 1;
00645 (thee->nion)++;
00646 thee->setnion = 1;
00647 return VRC_SUCCESS;
00648
00649 VERROR1:
00650     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00651     return VRC_FAILURE;
00652 }
00653
00654 VPRIPRIVATE int PBParm_parsePDIE(PBParm *thee, Vio *sock) {

```

```

00655     char tok[VMAX_BUFSIZE];
00656     double tf;
00657
00658     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00659     if (sscanf(tok, "%lf", &tf) == 0) {
00660         Vnm_print(2, "NOsh: Read non-float (%s) while parsing PDIE \
00661 keyword!\n", tok);
00662         return -1;
00663     }
00664     thee->pdie = tf;
00665     thee->setpdie = 1;
00666     return 1;
00667
00668     VERROR1:
00669         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00670         return -1;
00671 }
00672
00673 VPRIIVATE int PBEparm_parseSDENS(PBEparm *thee, Vio *sock) {
00674     char tok[VMAX_BUFSIZE];
00675     double tf;
00676
00677     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00678     if (sscanf(tok, "%lf", &tf) == 0) {
00679         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SDENS \
00680 keyword!\n", tok);
00681         return -1;
00682     }
00683     thee->sdens = tf;
00684     thee->setsdens = 1;
00685     return 1;
00686
00687     VERROR1:
00688         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00689         return -1;
00690 }
00691
00692 VPRIIVATE int PBEparm_parseSDIE(PBEparm *thee, Vio *sock) {
00693     char tok[VMAX_BUFSIZE];
00694     double tf;
00695
00696     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00697     if (sscanf(tok, "%lf", &tf) == 0) {
00698         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SDIE \
00699 keyword!\n", tok);
00700         return -1;
00701     }
00702     thee->sdie = tf;
00703     thee->setsdie = 1;
00704     return 1;
00705
00706     VERROR1:
00707         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00708         return -1;
00709 }
00710
00711 VPRIIVATE int PBEparm_parseSRFM(PBEparm *thee, Vio *sock) {

```

```

00712     char tok[VMAX_BUFSIZE];
00713     int ti;
00714
00715     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00716
00717     /* Parse old-style int arg */
00718     if (sscanf(tok, "%d", &ti) == 1) {
00719         thee->srfm = ti;
00720         thee->setsrfm = 1;
00721
00722         Vnm_print(2, "parsePBE: Warning -- parsed deprecated \"srfm %d\" \
00723 statement.\n", ti);
00724         Vnm_print(2, "parsePBE: Please use \"srfm \"");
00725         switch (thee->srfm) {
00726             case VSM_MOL:
00727                 Vnm_print(2, "mol");
00728                 break;
00729             case VSM_MOLSMOOTH:
00730                 Vnm_print(2, "smol");
00731                 break;
00732             case VSM_SPLINE:
00733                 Vnm_print(2, "spl2");
00734                 break;
00735             case VSM_SPLINE3:
00736                 Vnm_print(2, "spl3");
00737                 break;
00738             case VSM_SPLINE4:
00739                 Vnm_print(2, "spl4");
00740                 break;
00741             default:
00742                 Vnm_print(2, "UNKNOWN");
00743                 break;
00744         }
00745         Vnm_print(2, "\" instead.\n");
00746         return 1;
00747
00748     /* Parse newer text-based args */
00749     } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00750         thee->srfm = VSM_MOL;
00751         thee->setsrfm = 1;
00752         return 1;
00753     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
00754         thee->srfm = VSM_MOLSMOOTH;
00755         thee->setsrfm = 1;
00756         return 1;
00757     } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
00758         thee->srfm = VSM_SPLINE;
00759         thee->setsrfm = 1;
00760         return 1;
00761     } else if (Vstring_strcasecmp(tok, "spl3") == 0) {
00762         thee->srfm = VSM_SPLINE3;
00763         thee->setsrfm = 1;
00764         return 1;
00765     } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
00766         thee->srfm = VSM_SPLINE4;
00767         thee->setsrfm = 1;
00768         return 1;

```

```

00769     } else {
00770         Vnm_print(2, "NOSH: Unrecognized keyword (%s) when parsing \
00771 srfm!\n", tok);
00772         return -1;
00773     }
00774
00775     return 0;
00776
00777 VERROR1:
00778     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00779     return -1;
00780 }
00781
00782 VPRIIVATE int PBEparm_parseSRAD(PBEparm *thee, Vio *sock) {
00783     char tok[VMAX_BUFSIZE];
00784     double tf;
00785
00786     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00787     if (sscanf(tok, "%lf", &tf) == 0) {
00788         Vnm_print(2, "NOSH: Read non-float (%s) while parsing SRAD \
00789 keyword!\n", tok);
00790         return -1;
00791     }
00792     thee->srad = tf;
00793     thee->setssrad = 1;
00794     return 1;
00795
00796 VERROR1:
00797     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00798     return -1;
00799 }
00800
00801 VPRIIVATE int PBEparm_parseSWIN(PBEparm *thee, Vio *sock) {
00802     char tok[VMAX_BUFSIZE];
00803     double tf;
00804
00805     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00806     if (sscanf(tok, "%lf", &tf) == 0) {
00807         Vnm_print(2, "NOSH: Read non-float (%s) while parsing SWIN \
00808 keyword!\n", tok);
00809         return -1;
00810     }
00811     thee->swin = tf;
00812     thee->setswin = 1;
00813     return 1;
00814
00815 VERROR1:
00816     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00817     return -1;
00818 }
00819
00820 VPRIIVATE int PBEparm_parseTEMP(PBEparm *thee, Vio *sock) {
00821     char tok[VMAX_BUFSIZE];
00822     double tf;
00823
00824     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00825     if (sscanf(tok, "%lf", &tf) == 0) {

```

```

00826         Vnm_print(2, "NOsh: Read non-float (%s) while parsing TEMP \
00827 keyword!\n", tok);
00828         return -1;
00829     }
00830     thee->temp = tf;
00831     thee->settemp = 1;
00832     return 1;
00833
00834     VERROR1:
00835         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00836         return -1;
00837 }
00838
00839 VPRIVATE int PBParm_parseUSEMAP (PBParm *thee, Vio *sock) {
00840     char tok[VMAX_BUFSIZE];
00841     int ti;
00842
00843     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00844     Vnm_print(0, "PBParm_parseToken: Read %s...\n", tok);
00845     if (Vstring_strcasecmp(tok, "diel") == 0) {
00846         thee->useDielMap = 1;
00847         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00848         if (sscanf(tok, "%d", &ti) == 0) {
00849             Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00850 USEMAP DIEL keyword!\n", tok);
00851             return -1;
00852         }
00853         thee->dielMapID = ti;
00854         return 1;
00855     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
00856         thee->useKappaMap = 1;
00857         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00858         if (sscanf(tok, "%d", &ti) == 0) {
00859             Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00860 USEMAP KAPPA keyword!\n", tok);
00861             return -1;
00862         }
00863         thee->kappaMapID = ti;
00864         return 1;
00865     } else if (Vstring_strcasecmp(tok, "pot") == 0) {
00866         thee->usePotMap = 1;
00867         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00868         if (sscanf(tok, "%d", &ti) == 0) {
00869             Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00870 USEMAP POT keyword!\n", tok);
00871             return -1;
00872         }
00873         thee->potMapID = ti;
00874         return 1;
00875     } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00876         thee->useChargeMap = 1;
00877         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00878         if (sscanf(tok, "%d", &ti) == 0) {
00879             Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00880 USEMAP CHARGE keyword!\n", tok);
00881             return -1;
00882     }

```

```

00883     thee->chargeMapID = ti;
00884     return 1;
00885 } else {
00886     Vnm_print(2, "NOsh:  Read undefined keyword (%s) while parsing \
00887 USEMAP statement!\n", tok);
00888     return -1;
00889 }
00890 return 0;
00891
00892 VERROR1:
00893     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00894     return -1;
00895 }
00896
00897 VPRIVATE int PBEparm_parseCALCENERGY(PBEparm *thee, Vio *sock) {
00898     char tok[VMAX_BUFSIZE];
00899     int ti;
00900
00901     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00902     /* Parse number */
00903     if (sscanf(tok, "%d", &ti) == 1) {
00904         thee->calcenergy = ti;
00905         thee->setcalcenergy = 1;
00906
00907         Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"calcenergy \
00908 %d\" statement.\n", ti);
00909         Vnm_print(2, "parsePBE:  Please use \"calcenergy \"");
00910         switch (thee->calcenergy) {
00911             case PCE_NO:
00912                 Vnm_print(2, "no");
00913                 break;
00914             case PCE_TOTAL:
00915                 Vnm_print(2, "total");
00916                 break;
00917             case PCE_COMPS:
00918                 Vnm_print(2, "comps");
00919                 break;
00920             default:
00921                 Vnm_print(2, "UNKNOWN");
00922                 break;
00923         }
00924         Vnm_print(2, "\" instead.\n");
00925         return 1;
00926     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00927         thee->calcenergy = PCE_NO;
00928         thee->setcalcenergy = 1;
00929         return 1;
00930     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00931         thee->calcenergy = PCE_TOTAL;
00932         thee->setcalcenergy = 1;
00933         return 1;
00934     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00935         thee->calcenergy = PCE_COMPS;
00936         thee->setcalcenergy = 1;
00937         return 1;
00938     } else {
00939         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \

```

```

00940 calcenergy!\\n", tok);
00941     return -1;
00942 }
00943 return 0;
00944
00945 VERROR1:
00946     Vnm_print(2, "parsePBE: ran out of tokens!\\n");
00947     return -1;
00948 }
00949
00950 VPRIVATE int PB_parm_parseCALCFORCE(PB_parm *thee, Vio *sock) {
00951     char tok[VMAX_BUFSIZE];
00952     int ti;
00953
00954     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00955     /* Parse number */
00956     if (sscanf(tok, "%d", &ti) == 1) {
00957         thee->calcforce = ti;
00958         thee->setcalcforce = 1;
00959
00960         Vnm_print(2, "parsePBE: Warning -- parsed deprecated \\\"calcforce \\"
00961 %d\\\" statement.\\n", ti);
00962         Vnm_print(2, "parsePBE: Please use \\\"calcforce ");
00963         switch (thee->calcregy) {
00964             case PCF_NO:
00965                 Vnm_print(2, "no");
00966                 break;
00967             case PCF_TOTAL:
00968                 Vnm_print(2, "total");
00969                 break;
00970             case PCF_COMPS:
00971                 Vnm_print(2, "comps");
00972                 break;
00973             default:
00974                 Vnm_print(2, "UNKNOWN");
00975                 break;
00976         }
00977         Vnm_print(2, "\\\" instead.\\n");
00978         return 1;
00979     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00980         thee->calcforce = PCF_NO;
00981         thee->setcalcforce = 1;
00982         return 1;
00983     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00984         thee->calcforce = PCF_TOTAL;
00985         thee->setcalcforce = 1;
00986         return 1;
00987     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00988         thee->calcforce = PCF_COMPS;
00989         thee->setcalcforce = 1;
00990         return 1;
00991     } else {
00992         Vnm_print(2, "Nosh: Unrecognized parameter (%s) while parsing \\
00993 calcforce!\\n", tok);
00994         return -1;
00995     }
00996     return 0;

```

```

00997
00998     VERROR1:
00999         Vnm_print(2, "parsePBE: ran out of tokens!\n");
01000         return -1;
01001 }
01002
01003 /*-----*/
01004 /* Added by Michael Grabe */
01005 /*-----*/
01006
01007 VPRIVATE int PBEparm_parseZMEM(PBEparm *thee, Vio *sock) {
01008     char tok[VMAX_BUFSIZE];
01009     double tf;
01010
01011     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01012     if (sscanf(tok, "%lf", &tf) == 0) {
01013         Vnm_print(2, "NOsh: Read non-float (%s) while parsing ZMEM \
01014 keyword!\n", tok);
01015         return -1;
01016     }
01017     thee->zmem = tf;
01018     thee->setzmem = 1;
01019     return 1;
01020
01021 VERROR1:
01022     Vnm_print(2, "parsePBE: ran out of tokens!\n");
01023     return -1;
01024 }
01025
01026
01027 VPRIVATE int PBEparm_parseLMEM(PBEparm *thee, Vio *sock) {
01028     char tok[VMAX_BUFSIZE];
01029     double tf;
01030
01031     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01032     if (sscanf(tok, "%lf", &tf) == 0) {
01033         Vnm_print(2, "NOsh: Read non-float (%s) while parsing LMEM \
01034 keyword!\n", tok);
01035         return -1;
01036     }
01037     thee->Lmem = tf;
01038     thee->setLmem = 1;
01039     return 1;
01040
01041 VERROR1:
01042     Vnm_print(2, "parsePBE: ran out of tokens!\n");
01043     return -1;
01044 }
01045
01046 VPRIVATE int PBEparm_parseMDIE(PBEparm *thee, Vio *sock) {
01047     char tok[VMAX_BUFSIZE];
01048     double tf;
01049
01050     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01051     if (sscanf(tok, "%lf", &tf) == 0) {
01052         Vnm_print(2, "NOsh: Read non-float (%s) while parsing MDIE \
01053 keyword!\n", tok);

```

```

01054         return -1;
01055     }
01056     thee->mdie = tf;
01057     thee->setmdie = 1;
01058     return 1;
01059
01060 VERROR1:
01061 Vnm_print(2, "parsePBE: ran out of tokens!\n");
01062     return -1;
01063 }
01064
01065 VPRIVATE int PBParm_parseMEMV(PBParm *thee, Vio *sock) {
01066     char tok[VMAX_BUFSIZE];
01067     double tf;
01068
01069     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01070     if (sscanf(tok, "%lf", &tf) == 0) {
01071         Vnm_print(2, "NOSh: Read non-float (%s) while parsing MEMV \
01072 keyword!\n", tok);
01073         return -1;
01074     }
01075     thee->memv = tf;
01076     thee->setmemv = 1;
01077     return 1;
01078
01079 VERROR1:
01080 Vnm_print(2, "parsePBE: ran out of tokens!\n");
01081     return -1;
01082 }
01083
01084 /*-----*/
01085
01086 VPRIVATE int PBParm_parseWRITE(PBParm *thee, Vio *sock) {
01087     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
01088     Vdata_Type writetype;
01089     Vdata_Format writefmt;
01090
01091     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01092     if (Vstring_strcasecmp(tok, "pot") == 0) {
01093         writetype = VDT_POT;
01094     } else if (Vstring_strcasecmp(tok, "atompot") == 0) {
01095         writetype = VDT_ATOMPOT;
01096     } else if (Vstring_strcasecmp(tok, "charge") == 0) {
01097         writetype = VDT_CHARGE;
01098     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
01099         writetype = VDT_SMOL;
01100     } else if (Vstring_strcasecmp(tok, "dielx") == 0) {
01101         writetype = VDT_DIELX;
01102     } else if (Vstring_strcasecmp(tok, "diely") == 0) {
01103         writetype = VDT_DIELY;
01104     } else if (Vstring_strcasecmp(tok, "dielz") == 0) {
01105         writetype = VDT_DIELZ;
01106     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
01107         writetype = VDT_KAPPA;
01108     } else if (Vstring_strcasecmp(tok, "sspl") == 0) {
01109         writetype = VDT_SSPL;
01110     } else if (Vstring_strcasecmp(tok, "vdw") == 0) {

```

```

01111     writetype = VDT_VDW;
01112 } else if (Vstring_strcasecmp(tok, "ivdw") == 0) {
01113     writetype = VDT_IVDW;
01114 } else if (Vstring_strcasecmp(tok, "lap") == 0) {
01115     writetype = VDT_LAP;
01116 } else if (Vstring_strcasecmp(tok, "edens") == 0) {
01117     writetype = VDT_EDENS;
01118 } else if (Vstring_strcasecmp(tok, "ndens") == 0) {
01119     writetype = VDT_NDENS;
01120 } else if (Vstring_strcasecmp(tok, "qdens") == 0) {
01121     writetype = VDT_QDENS;
01122 } else {
01123     Vnm_print(2, "PBEparm_parse: Invalid data type (%s) to write!\n",
01124             tok);
01125     return -1;
01126 }
01127 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01128 if (Vstring_strcasecmp(tok, "dx") == 0) {
01129     writefmt = VDF_DX;
01130 } else if (Vstring_strcasecmp(tok, "uhbd") == 0) {
01131     writefmt = VDF_UHBD;
01132 } else if (Vstring_strcasecmp(tok, "avs") == 0) {
01133     writefmt = VDF_AVF;
01134 } else if (Vstring_strcasecmp(tok, "gz") == 0) {
01135     writefmt = VDF_GZ;
01136 } else if (Vstring_strcasecmp(tok, "flat") == 0) {
01137     writefmt = VDF_FLAT;
01138 } else {
01139     Vnm_print(2, "PBEparm_parse: Invalid data format (%s) to write!\n",
01140             tok);
01141     return -1;
01142 }
01143 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01144 if (tok[0]=='') {
01145     strcpy(strnew, "");
01146     while (tok[strlen(tok)-1] != '') {
01147         strcat(str, tok);
01148         strcat(str, " ");
01149         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01150     }
01151     strcat(str, tok);
01152     strncpy(strnew, str+1, strlen(str)-2);
01153     strcpy(tok, strnew);
01154 }
01155 if (thee->numwrite < (PBEPARM_MAXWRITE-1)) {
01156     strncpy(thee->writestem[thee->numwrite], tok, VMAX_ARGLEN);
01157     thee->writetype[thee->numwrite] = writetype;
01158     thee->writefmt[thee->numwrite] = writefmt;
01159     (thee->numwrite)++;
01160 } else {
01161     Vnm_print(2, "PBEparm_parse: You have exceeded the maximum number of write sta-
01162     tements!\n");
01163 }
01164     return 1;
01165
01166 VERROR1:

```

```

01167         Vnm_print(2, "parsePBE: ran out of tokens!\n");
01168         return -1;
01169     }
01170
01171 VPRIVATE int PBEParm_parseWITEMAT(PBEParm *thee, Vio *sock) {
01172     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
01173
01174     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01175     if (Vstring_strcasecmp(tok, "poisson") == 0) {
01176         thee->witematflag = 0;
01177     } else if (Vstring_strcasecmp(tok, "full") == 0) {
01178         thee->witematflag = 1;
01179     } else {
01180         Vnm_print(2, "NOsh: Invalid format (%s) while parsing \
01181 WITEMAT keyword!\n", tok);
01182         return -1;
01183     }
01184
01185     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01186     if (tok[0]=='/') {
01187         strcpy(strnew, "");
01188         while (tok[strlen(tok)-1] != '/') {
01189             strcat(str, tok);
01190             strcat(str, " ");
01191             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01192         }
01193         strcat(str, tok);
01194         strncpy(strnew, str+1, strlen(str)-2);
01195         strcpy(tok, strnew);
01196     }
01197     strncpy(thee->witematstem, tok, VMAX_ARGLEN);
01198     thee->setwitemat = 1;
01199     thee->witemat = 1;
01200     return 1;
01201
01202     VERROR1:
01203         Vnm_print(2, "parsePBE: ran out of tokens!\n");
01204         return -1;
01205
01206 }
01207
01208 VPUBLIC int PBEParm_parseToken(PBEParm *thee, char tok[VMAX_BUFSIZE],
01209     Vio *sock) {
01210
01211     if (thee == VNULL) {
01212         Vnm_print(2, "parsePBE: got NULL thee!\n");
01213         return -1;
01214     }
01215     if (sock == VNULL) {
01216         Vnm_print(2, "parsePBE: got NULL socket!\n");
01217         return -1;
01218     }
01219
01220     Vnm_print(0, "PBEParm_parseToken: trying %s...\n", tok);
01221
01222     if (Vstring_strcasecmp(tok, "mol") == 0) {
01223         return PBEParm_parseMOL(thee, sock);

```

```

01224     } else if (Vstring_strcasecmp(tok, "lpbe") == 0) {
01225         return PBEparm_parseLPBE(thee, sock);
01226     } else if (Vstring_strcasecmp(tok, "npbe") == 0) {
01227         return PBEparm_parseNPBE(thee, sock);
01228     } else if (Vstring_strcasecmp(tok, "lrbpe") == 0) {
01229         return PBEparm_parseLRPBE(thee, sock);
01230     } else if (Vstring_strcasecmp(tok, "nrpb") == 0) {
01231         return PBEparm_parseNRPBE(thee, sock);
01232     } else if (Vstring_strcasecmp(tok, "smpbe") == 0) {
01233         return PBEparm_parseSMPBE(thee, sock);
01234     } else if (Vstring_strcasecmp(tok, "bcfl") == 0) {
01235         return PBEparm_parseBCFL(thee, sock);
01236     } else if (Vstring_strcasecmp(tok, "ion") == 0) {
01237         return PBEparm_parseION(thee, sock);
01238     } else if (Vstring_strcasecmp(tok, "pdie") == 0) {
01239         return PBEparm_parsePDIE(thee, sock);
01240     } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
01241         return PBEparm_parseSDENS(thee, sock);
01242     } else if (Vstring_strcasecmp(tok, "sdie") == 0) {
01243         return PBEparm_parseSDIE(thee, sock);
01244     } else if (Vstring_strcasecmp(tok, "srdfm") == 0) {
01245         return PBEparm_parseSRFM(thee, sock);
01246     } else if (Vstring_strcasecmp(tok, "srad") == 0) {
01247         return PBEparm_parseSRAD(thee, sock);
01248     } else if (Vstring_strcasecmp(tok, "swin") == 0) {
01249         return PBEparm_parseSWIN(thee, sock);
01250     } else if (Vstring_strcasecmp(tok, "temp") == 0) {
01251         return PBEparm_parseTEMP(thee, sock);
01252     } else if (Vstring_strcasecmp(tok, "usemap") == 0) {
01253         return PBEparm_parseUSEMAP(thee, sock);
01254     } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
01255         return PBEparm_parseCALCENERGY(thee, sock);
01256     } else if (Vstring_strcasecmp(tok, "calccforce") == 0) {
01257         return PBEparm_parseCALCFORCE(thee, sock);
01258     } else if (Vstring_strcasecmp(tok, "write") == 0) {
01259         return PBEparm_parseWRITE(thee, sock);
01260     } else if (Vstring_strcasecmp(tok, "writemat") == 0) {
01261         return PBEparm_parseWRITEMAT(thee, sock);
01262
01263     /*-----*/
01264     /* Added by Michael Grabe */
01265     /*-----*/
01266
01267     } else if (Vstring_strcasecmp(tok, "zmem") == 0) {
01268         return PBEparm_parseZMEM(thee, sock);
01269     } else if (Vstring_strcasecmp(tok, "Lmem") == 0) {
01270         return PBEparm_parseLMEM(thee, sock);
01271     } else if (Vstring_strcasecmp(tok, "mdie") == 0) {
01272         return PBEparm_parseMDIE(thee, sock);
01273     } else if (Vstring_strcasecmp(tok, "memv") == 0) {
01274         return PBEparm_parseMEMV(thee, sock);
01275     }
01276
01277     /*-----*/
01278
01279     return 0;
01280

```

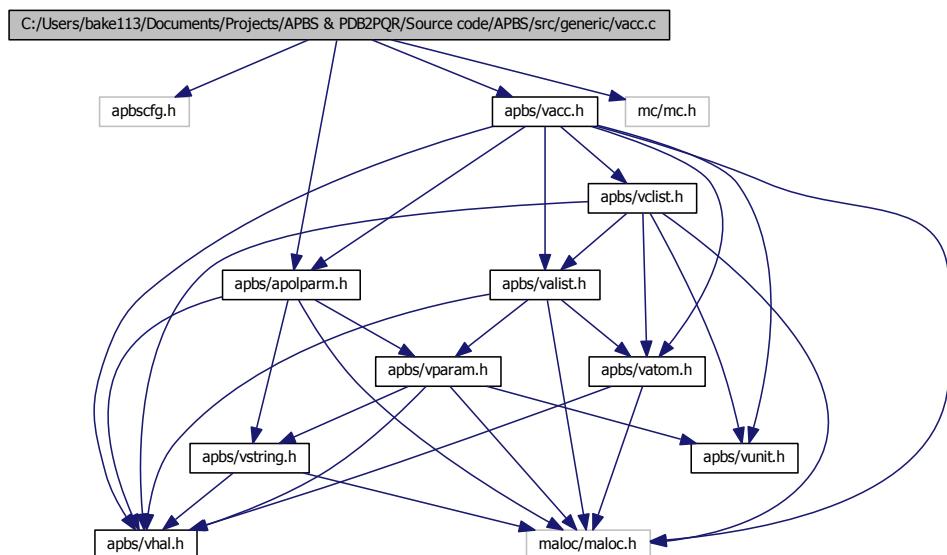
```
01281 }
```

10.61 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vacc.c File Reference

Class Vacc methods.

```
#include "apbscfg.h"  
#include "apbs/vacc.h"  
#include "apbs/apolparm.h"  
#include "mc/mc.h"
```

Include dependency graph for vacc.c:



Functions

- VPUBLIC unsigned long int [Vacc_memChk](#) (Vacc *thee)
Get number of bytes in this object and its members.

- VPRIVATE int `ivdwAccExclus` (`Vacc` *thee, double center[3], double radius, int atomID)

Determines if a point is within the union of the spheres centered at the atomic centers with radii equal to the sum of their van der Waals radii and the probe radius. Does not include contributions from the specified atom.
- VPUBLIC `Vacc *` `Vacc_ctor` (`Valist` *alist, `Vclist` *clist, double surf_density)

Construct the accessibility object.
- VPRIVATE int `Vacc_storeParms` (`Vacc` *thee, `Valist` *alist, `Vclist` *clist, double surf_density)
- VPRIVATE int `Vacc_allocate` (`Vacc` *thee)
- VPUBLIC int `Vacc_ctor2` (`Vacc` *thee, `Valist` *alist, `Vclist` *clist, double surf_density)

FORTRAN stub to construct the accessibility object.
- VPUBLIC void `Vacc_dtor` (`Vacc` **thee)

Destroy object.
- VPUBLIC void `Vacc_dtor2` (`Vacc` *thee)

FORTRAN stub to destroy object.
- VPUBLIC double `Vacc_vdwAcc` (`Vacc` *thee, double center[3])
- VPUBLIC double `Vacc_ivdwAcc` (`Vacc` *thee, double center[3], double radius)
- VPUBLIC void `Vacc_splineAccGradAtomNorm` (`Vacc` *thee, double center[VAPBS_DIM], double win, double infrad, `Vatom` *atom, double *grad)

Report gradient of spline-based accessibility with respect to a particular atom normalized by the accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)
- VPUBLIC void `Vacc_splineAccGradAtomUnnorm` (`Vacc` *thee, double center[VAPBS_DIM], double win, double infrad, `Vatom` *atom, double *grad)

Report gradient of spline-based accessibility with respect to a particular atom (see `Vpmg_splineAccAtom`)
- VPUBLIC double `Vacc_splineAccAtom` (`Vacc` *thee, double center[VAPBS_DIM], double win, double infrad, `Vatom` *atom)

Report spline-based accessibility for a given atom.
- VPRIVATE double `splineAcc` (`Vacc` *thee, double center[VAPBS_DIM], double win, double infrad, `VclistCell` *cell)

Fast spline-based surface computation subroutine.
- VPUBLIC double `Vacc_splineAcc` (`Vacc` *thee, double center[VAPBS_DIM], double win, double infrad)

Report spline-based accessibility.
- VPUBLIC void `Vacc_splineAccGrad` (`Vacc` *thee, double center[VAPBS_DIM], double win, double infrad, double *grad)

Report gradient of spline-based accessibility.
- VPUBLIC double `Vacc_molAcc` (`Vacc` *thee, double center[VAPBS_DIM], double radius)

Report molecular accessibility.

- VPUBLIC double `Vacc_fastMolAcc` (`Vacc *thee`, double `center[VAPBS_DIM]`, double `radius`)
Report molecular accessibility quickly.
- VPUBLIC void `Vacc_writeGMV` (`Vacc *thee`, double `radius`, int `meth`, `Gem *gm`, char `*iodev`, char `*iofmt`, char `*iohost`, char `*iofile`)
- VPUBLIC double `Vacc_SASA` (`Vacc *thee`, double `radius`)
Build the solvent accessible surface (SAS) and calculate the solvent accessible surface area.
- VPUBLIC double `Vacc_totalSASA` (`Vacc *thee`, double `radius`)
Return the total solvent accessible surface area (SASA)
- VPUBLIC double `Vacc_atomSASA` (`Vacc *thee`, double `radius`, `Vatom *atom`)
Return the atomic solvent accessible surface area (SASA)
- VPUBLIC `VaccSurf * VaccSurf_ctor` (`Vmem *mem`, double `probe_radius`, int `nsphere`)
Allocate and construct the surface object; do not assign surface points to positions.
- VPUBLIC int `VaccSurf_ctor2` (`VaccSurf *thee`, `Vmem *mem`, double `probe_radius`, int `nsphere`)
Construct the surface object using previously allocated memory; do not assign surface points to positions.
- VPUBLIC void `VaccSurf_dtor` (`VaccSurf **thee`)
Destroy the surface object and free its memory.
- VPUBLIC void `VaccSurf_dtor2` (`VaccSurf *thee`)
Destroy the surface object.
- VPUBLIC `VaccSurf * Vacc_atomSurf` (`Vacc *thee`, `Vatom *atom`, `VaccSurf *ref`, double `prad`)
Set up an array of points corresponding to the SAS due to a particular atom.
- VPUBLIC `VaccSurf * VaccSurf_refSphere` (`Vmem *mem`, int `npts`)
Set up an array of points for a reference sphere of unit radius.
- VPUBLIC `VaccSurf * Vacc_atomSASPoints` (`Vacc *thee`, double `radius`, `Vatom *atom`)
Get the set of points for this atom's solvent-accessible surface.
- VPUBLIC void `Vacc_splineAccGradAtomNorm4` (`Vacc *thee`, double `center[VAPBS_DIM]`, double `win`, double `infrad`, `Vatom *atom`, double `*grad`)
Report gradient of spline-based accessibility with respect to a particular atom normalized by a 4th order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)
- VPUBLIC void `Vacc_splineAccGradAtomNorm3` (`Vacc *thee`, double `center[VAPBS_DIM]`, double `win`, double `infrad`, `Vatom *atom`, double `*grad`)
Report gradient of spline-based accessibility with respect to a particular atom normalized by a 3rd order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)

- VPUBLIC void `Vacc_atomdSAV` (`Vacc *thee`, double `srad`, `Vatom *atom`, double `*dSA`)

Get the derivative of solvent accessible volume.
- VPRIVATE double `Vacc_SASAPOS` (`Vacc *thee`, double `radius`)
- VPRIVATE double `Vacc_atomSASAPOS` (`Vacc *thee`, double `radius`, `Vatom *atom`, int `mode`)
- VPUBLIC void `Vacc_atomdSASA` (`Vacc *thee`, double `dpos`, double `srad`, `Vatom *atom`, double `*dSA`)

Get the derivative of solvent accessible area.
- VPUBLIC void `Vacc_totalAtomdSASA` (`Vacc *thee`, double `dpos`, double `srad`, `Vatom *atom`, double `*dSA`)

Testing purposes only.
- VPUBLIC void `Vacc_totalAtomdSAV` (`Vacc *thee`, double `dpos`, double `srad`, `Vatom *atom`, double `*dSA`, `Vclist *clist`)

Total solvent accessible volume.
- VPUBLIC double `Vacc_totalsAV` (`Vacc *thee`, `Vclist *clist`, `APOLparm *apolparm`, double `radius`)

Return the total solvent accessible volume (SAV)
- int `Vacc_wcaEnergyAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`, int `iatom`, double `*value`)

Calculate the WCA energy for an atom.
- VPUBLIC int `Vacc_wcaEnergy` (`Vacc *acc`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`)

Return the WCA integral energy.
- VPUBLIC int `Vacc_wcaForceAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Vclist *clist`, `Vatom *atom`, double `*force`)

Return the WCA integral force.

10.61.1 Detailed Description

Class `Vacc` methods.

Author

Nathan Baker

Version

Id:

`vacc.c` 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [vacc.c](#).

10.61.2 Function Documentation

```
10.61.2.1 VPRIVATE int ivdwAccExclus ( Vacc *thee, double center[3], double radius, int atomID )
```

Determines if a point is within the union of the spheres centered at the atomic centers with radii equal to the sum of their van der Waals radii and the probe radius. Does not include contributions from the specified atom.

Returns

1 if accessible (outside the inflated van der Waals radius), 0 otherwise

Author

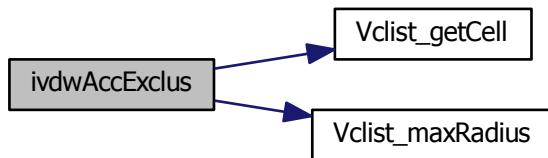
Nathan Baker

Parameters

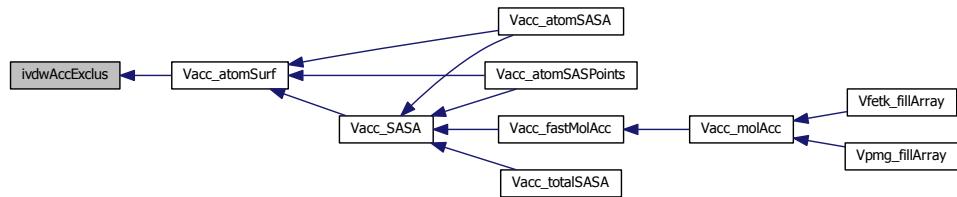
<i>center</i>	Accessibility object
<i>radius</i>	Position to test
<i>atomID</i>	Radius of probe ID of atom to ignore

Definition at line 85 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



10.61.2.2 VPRIvate double splineAcc (*Vacc * thee, double center[VAPBS_DIM], double win, double infrad, VclistCell * cell*)

Fast spline-based surface computation subroutine.

Returns

Spline value

Author

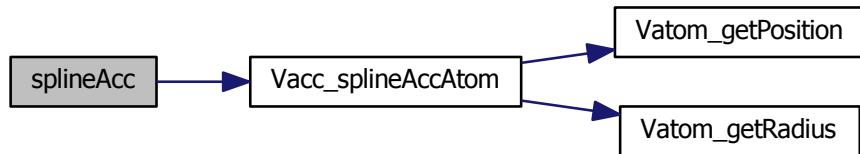
Todd Dolinsky and Nathan Baker

Parameters

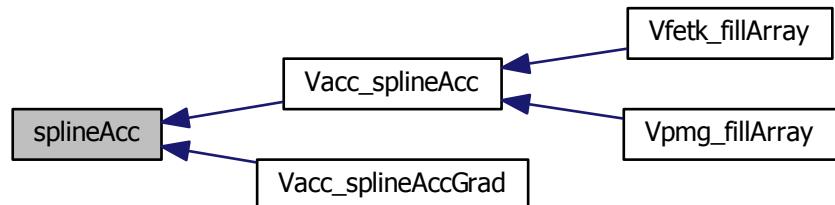
<i>center</i>	Accessibility object
<i>win</i>	Point at which the acc is to be evaluated
<i>infrad</i>	Spline window
<i>cell</i>	Radius to inflate atomic radius Cell of atom objects

Definition at line [437](#) of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



10.61.2.3 VPRIVATE int Vacc_allocate (Vacc * thee)

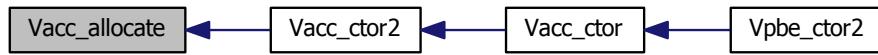
Allocate (and clear) space for storage

Definition at line 184 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

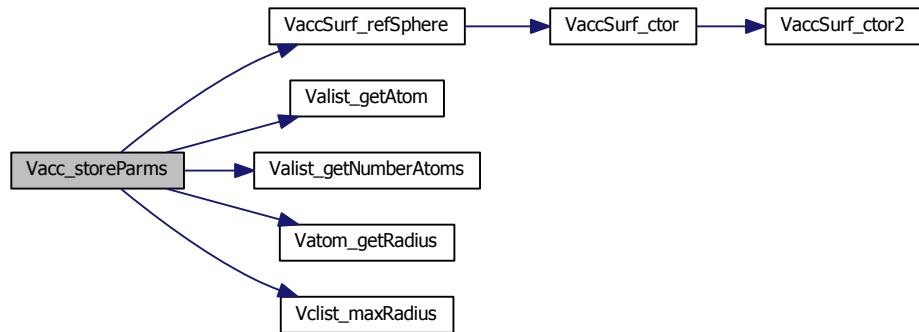


10.61.2.4 VPRIATE int Vacc_storeParms (*Vacc *thee*, *Valist *alist*, *Vclist *clist*, double *surf_density*)

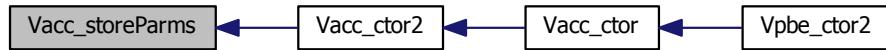
Check and store parameters passed to constructor

Definition at line 145 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



10.62 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vacc.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vacc.h"
00059 #include "apbs/apolparm.h"
00060
00061 #if defined(HAVE_MC_H)
00062 #include "mc/mc.h"
00063 #endif
00064
00065 VEMBED(rcsid="$Id: vacc.c 1667 2011-12-02 23:22:02Z pcellis $")
00066
00067 #if !defined(VINLINE_VACC)
  
```

```

00068
00069 VPUBLIC unsigned long int Vacc_memChk(Vacc *thee) {
00070     if (thee == VNULL) return 0;
00071     return Vmem_bytes(thee->mem);
00072 }
00073
00074 #endif /* if !defined(VINLINE_VACC) */
00075
00076 VPUBLIC int ivdwAccExclus(
00077     Vacc *thee,
00078     double center[3],
00079     double radius,
00080     int atomID
00081     ) {
00082
00083     int iatom;
00084     double dist2, *apos;
00085     Vatom *atom;
00086     VclistCell *cell;
00087
00088     VASSERT(thee != VNULL);
00089
00090     /* We can only test probes with radii less than the max specified */
00091     if (radius > Vclist_maxRadius(thee->clist)) {
00092         Vnm_print(2,
00093             "Vacc_ivdwAcc: got radius (%g) bigger than max radius (%g)\n",
00094             radius, Vclist_maxRadius(thee->clist));
00095     }
00096     VASSERT(0);
00097
00098     /* Get the relevant cell from the cell list */
00099     cell = Vclist_getCell(thee->clist, center);
00100
00101     /* If we have no cell, then no atoms are nearby and we're definitely
00102      * accessible */
00103     if (cell == VNULL) {
00104         return 1;
00105     }
00106
00107     /* Otherwise, check for overlap with the atoms in the cell */
00108     for (iatom=0; iatom<cell->natoms; iatom++) {
00109         atom = cell->atoms[iatom];
00110         apos = atom->position;
00111         dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00112             + VSQR(center[2]-apos[2]);
00113         if (dist2 < VSQR(atom->radius+radius)){
00114             if (atom->id != atomID) return 0;
00115         }
00116     }
00117
00118     /* If we're still here, then the point is accessible */
00119     return 1;
00120
00121 }
00122
00123 VPUBLIC Vacc* Vacc_ctor(Valist *alist, Vclist *clist, double surf_density) {
00124
00125 }
```

```

00134
00135     Vacc *thee = VNULL;
00136
00137     /* Set up the structure */
00138     thee = Vmem_malloc(VNULL, 1, sizeof(Vacc) );
00139     VASSERT( thee != VNULL);
00140     VASSERT( Vacc_ctor2(thee, alist, clist, surf_density));
00141     return thee;
00142 }
00143
00144 VPRIVATE int Vacc_storeParms(Vacc *thee, Valist *alist, Vclist *clist,
00145                               double surf_density) {
00146
00147     int nsphere, iatom;
00148     double maxrad, maxarea, rad;
00149     Vatom *atom;
00150
00151     if (alist == VNULL) {
00152         Vnm_print(2, "Vacc_storeParms: Got NULL Valist!\n");
00153         return 0;
00154     } else thee->alist = alist;
00155     if (clist == VNULL) {
00156         Vnm_print(2, "Vacc_storeParms: Got NULL Vclist!\n");
00157         return 0;
00158     } else thee->clist = clist;
00159     thee->surf_density = surf_density;
00160
00161     /* Loop through the atoms to determine the maximum radius */
00162     maxrad = 0.0;
00163     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00164         atom = Valist_getAtom(alist, iatom);
00165         rad = Vatom_getRadius(atom);
00166         if (rad > maxrad) maxrad = rad;
00167     }
00168     maxrad = maxrad + Vclist_maxRadius(thee->clist);
00169
00170     maxarea = 4.0*VPI*maxrad*maxrad;
00171     nsphere = (int)ceil(maxarea*surf_density);
00172
00173     Vnm_print(0, "Vacc_storeParms: Surf. density = %g\n", surf_density);
00174     Vnm_print(0, "Vacc_storeParms: Max area = %g\n", maxarea);
00175     thee->refSphere = VaccSurf_refSphere(thee->mem, nsphere);
00176     Vnm_print(0, "Vacc_storeParms: Using %d-point reference sphere\n",
00177               thee->refSphere->npts);
00178
00179     return 1;
00180 }
00181 }
00182
00183 VPRIVATE int Vacc_allocate(Vacc *thee) {
00184
00185     int i, natoms;
00186
00187     natoms = Valist_getNumberAtoms(thee->alist);
00188
00189     thee->atomFlags = Vmem_malloc(thee->mem, natoms, sizeof(int));
00190     if (thee->atomFlags == VNULL) {
00191         Vnm_print(2,

```

```

00193             "Vacc_allocate: Failed to allocate %d (int)s for atomFlags!\n",
00194             natoms);
00195         return 0;
00196     }
00197     for (i=0; i<natoms; i++) (thee->atomFlags)[i] = 0;
00198
00199     return 1;
00200 }
00201
00202
00203 VPUBLIC int Vacc_ctor2(Vacc *thee, Valist *alist, Vclist *clist,
00204     double surf_density) {
00205
00206     /* Check and store parameters */
00207     if (!Vacc_storeParms(thee, alist, clist, surf_density)) {
00208         Vnm_print(2, "Vacc_ctor2: parameter check failed!\n");
00209         return 0;
00210     }
00211
00212     /* Set up memory management object */
00213     thee->mem = Vmem_ctor("APBS::VACC");
00214     if (thee->mem == VNULL) {
00215         Vnm_print(2, "Vacc_ctor2: memory object setup failed!\n");
00216         return 0;
00217     }
00218
00219     /* Setup and check probe */
00220     thee->surf = VNULL;
00221
00222     /* Allocate space */
00223     if (!Vacc_allocate(thee)) {
00224         Vnm_print(2, "Vacc_ctor2: memory allocation failed!\n");
00225         return 0;
00226     }
00227
00228     return 1;
00229 }
00230
00231
00232 VPUBLIC void Vacc_dtors(Vacc **thee) {
00233
00234     if ((*thee) != VNULL) {
00235         Vacc_dtors(*thee);
00236         Vmem_free(VNULL, 1, sizeof(Vacc), (void **)thee);
00237         (*thee) = VNULL;
00238     }
00239
00240 }
00241
00242 VPUBLIC void Vacc_dtors2(Vacc *thee) {
00243
00244     int i, natoms;
00245
00246     natoms = Valist_getNumberAtoms(thee->alist);
00247     Vmem_free(thee->mem, natoms, sizeof(int), (void **)&(thee->atomFlags));
00248
00249     if (thee->refSphere != VNULL) {

```

```

00250         VaccSurf_dtor(&(thee->refSphere));
00251         thee->refSphere = VNULL;
00252     }
00253     if (thee->surf != VNULL) {
00254         for (i=0; i<natoms; i++) VaccSurf_dtor(&(thee->surf[i]));
00255         Vmem_free(thee->mem, natoms, sizeof(VaccSurf *),
00256                   (void **) &(thee->surf));
00257         thee->surf = VNULL;
00258     }
00259     Vmem_dtor(&(thee->mem));
00260 }
00261 }
00262
00263 VPUBLIC double Vacc_vdwAcc(Vacc *thee, double center[3]) {
00264
00265     VclistCell *cell;
00266     Vatom *atom;
00267     int iatom;
00268     double *apos;
00269     double dist2;
00270
00271     /* Get the relevant cell from the cell list */
00272     cell = Vclist_getCell(thee->clist, center);
00273
00274     /* If we have no cell, then no atoms are nearby and we're definitely
00275      * accessible */
00276     if (cell == VNULL) return 1.0;
00277
00278     /* Otherwise, check for overlap with the atoms in the cell */
00279     for (iatom=0; iatom<cell->natoms; iatom++) {
00280         atom = cell->atoms[iatom];
00281         apos = Vatom_getPosition(atom);
00282         dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00283             + VSQR(center[2]-apos[2]);
00284         if (dist2 < VSQR(Vatom_getRadius(atom))) return 0.0;
00285     }
00286
00287     /* If we're still here, then the point is accessible */
00288     return 1.0;
00289 }
00290
00291 VPUBLIC double Vacc_ivdwAcc(Vacc *thee, double center[3], double radius) {
00292
00293     return (double)ivdwAccExclus(thee, center, radius, -1);
00294
00295 }
00296
00297 VPUBLIC void Vacc_splineAccGradAtomNorm(Vacc *thee, double center[VAPBS_DIM],
00298     double win, double infrad, Vatom *atom, double *grad) {
00299
00300     int i;
00301     double dist, *apos, arad, sm, sm2, w2i, w3i, mygrad;
00302     double mychi = 1.0;           /* Char. func. value for given atom */
00303
00304     VASSERT(thee != NULL);
00305
00306     /* Inverse squared window parameter */

```

```

00307     w2i = 1.0/(win*win);
00308     w3i = 1.0/(win*win*win);
00309
00310     /* The grad is zero by default */
00311     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00312
00313     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00314      * *** MAGNITUDE OF THE FORCE *** */
00315     apos = Vatom_getPosition(atom);
00316     /* Zero-radius atoms don't contribute */
00317     if (Vatom_getRadius(atom) > 0.0) {
00318         arad = Vatom_getRadius(atom) + inftrad;
00319         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00320                     + VSQR(apos[2]-center[2]));
00321         /* If we're inside an atom, the entire characteristic function
00322          * will be zero and the grad will be zero, so we can stop */
00323         if (dist < (arad - win)) return;
00324         /* Likewise, if we're outside the smoothing window, the characteristic
00325          * function is unity and the grad will be zero, so we can stop */
00326         else if (dist > (arad + win)) return;
00327         /* Account for floating point error at the border
00328          * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
00329          * (Vacc_splineAccAtom)? */
00330         else if ((VABS(dist - (arad - win)) < VSMALL) ||
00331                  (VABS(dist - (arad + win)) < VSMALL)) return;
00332         /* If we're inside the smoothing window */
00333         else {
00334             sm = dist - arad + win;
00335             sm2 = VSQR(sm);
00336             mychi = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00337             mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00338         }
00339         /* Now assemble the grad vector */
00340         VASSERT(mychi > 0.0);
00341         for (i=0; i<VAPBS_DIM; i++)
00342             grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
00343     }
00344 }
00345
00346 VPUBLIC void Vacc_splineAccGradAtomUnnorm(Vacc *thee, double center[VAPBS_DIM],
00347                                              double win, double inftrad, Vatom *atom, double *grad) {
00348
00349     int i;
00350     double dist, *apos, arad, sm, sm2, w2i, w3i, mygrad;
00351     double mychi = 1.0;           /* Char. func. value for given atom */
00352
00353     VASSERT(thee != NULL);
00354
00355     /* Inverse squared window parameter */
00356     w2i = 1.0/(win*win);
00357     w3i = 1.0/(win*win*win);
00358
00359     /* The grad is zero by default */
00360     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00361
00362     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00363      * *** MAGNITUDE OF THE FORCE *** */

```

```

00364     apos = Vatom_getPosition(atom);
00365     /* Zero-radius atoms don't contribute */
00366     if (Vatom_getRadius(atom) > 0.0) {
00367         arad = Vatom_getRadius(atom) + infrad;
00368         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00369                     + VSQR(apos[2]-center[2]));
00370         /* If we're inside an atom, the entire characteristic function
00371          * will be zero and the grad will be zero, so we can stop */
00372         if (dist < (arad - win)) return;
00373         /* Likewise, if we're outside the smoothing window, the characteristic
00374          * function is unity and the grad will be zero, so we can stop */
00375         else if (dist > (arad + win)) return;
00376         /* Account for floating point error at the border
00377          * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
00378          * (Vacc_splineAccAtom)? */
00379         else if ((VABS(dist - (arad - win)) < VSMALL) ||
00380                  (VABS(dist - (arad + win)) < VSMALL)) return;
00381         /* If we're inside the smoothing window */
00382         else {
00383             sm = dist - arad + win;
00384             sm2 = VSQR(sm);
00385             mychi = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00386             mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00387         }
00388         /* Now assemble the grad vector */
00389         VASSERT(mychi > 0.0);
00390         for (i=0; i<VAPBS_DIM; i++)
00391             grad[i] = -(mygrad)*((center[i] - apos[i])/dist);
00392     }
00393 }
00394
00395 VPUBLIC double Vacc_splineAccAtom(Vacc *thee, double center[VAPBS_DIM],
00396                                     double win, double infrad, Vatom *atom) {
00397
00398     double dist, *apos, arad, sm, sm2, w2i, w3i, value, stot, sctot;
00399
00400     VASSERT(thee != NULL);
00401
00402     /* Inverse squared window parameter */
00403     w2i = 1.0/(win*win);
00404     w3i = 1.0/(win*win*win);
00405
00406     apos = Vatom_getPosition(atom);
00407     /* Zero-radius atoms don't contribute */
00408     if (Vatom_getRadius(atom) > 0.0) {
00409         arad = Vatom_getRadius(atom) + infrad;
00410         stot = arad + win;
00411         sctot = VMAX2(0, (arad - win));
00412         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00413                     + VSQR(apos[2]-center[2]));
00414         /* If we're inside an atom, the entire characteristic function
00415          * will be zero */
00416         if ((dist < sctot) || (VABS(dist - sctot) < VSMALL)) {
00417             value = 0.0;
00418             /* We're outside the smoothing window */
00419             } else if ((dist > stot) || (VABS(dist - stot) < VSMALL)) {
00420                 value = 1.0;

```

```

00421     /* We're inside the smoothing window */
00422     } else {
00423         sm = dist - arad + win;
00424         sm2 = VSQR(sm);
00425         value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00426     }
00427 } else value = 1.0;
00428
00429     return value;
00430 }
00431
00432 VPRIIVATE double splineAcc(
00433     Vacc *thee,
00434     double center[VAPBS_DIM],
00435     double win,
00436     double infrad,
00437     VclistCell *cell
00438 ) {
00439
00440     int atomID, iatom;
00441     Vatom *atom;
00442     double value = 1.0;
00443
00444     VASSERT(thee != NULL);
00445
00446     /* Now loop through the atoms assembling the characteristic function */
00447     for (iatom=0; iatom<cell->natoms; iatom++) {
00448
00449         atom = cell->atoms[iatom];
00450         atomID = atom->id;
00451
00452         /* Check to see if we've counted this atom already */
00453         if ( !(thee->atomFlags[atomID]) ) {
00454
00455             thee->atomFlags[atomID] = 1;
00456             value *= Vacc_splineAccAtom(thee, center, win, infrad, atom);
00457
00458             if (value < VSMALL) return value;
00459         }
00460     }
00461
00462     return value;
00463 }
00464
00465
00466
00467
00468
00469 }
00470
00471
00472 VPUBLIC double Vacc_splineAcc(Vacc *thee, double center[VAPBS_DIM], double win,
00473     double infrad) {
00474
00475     VclistCell *cell;
00476     Vatom *atom;
00477     int iatom, atomID;
00478
00479     VASSERT(thee != NULL);
00480
00481     if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00482         Vnm_print(2, "Vacc_splineAcc: Vclist has max_radius=%g;\n",
00483

```

```

00484         Vclist_maxRadius(thee->clist));
00485         Vnm_print(2, "Vacc_splineAcc: Insufficient for win=%g, infrad=%g\n",
00486                     win, infrad);
00487         VASSERT(0);
00488     }
00489
00490     /* Get a cell or VNULL; in the latter case return 1.0 */
00491     cell = Vclist_getCell(thee->clist, center);
00492     if (cell == VNULL) return 1.0;
00493
00494     /* First, reset the list of atom flags
00495      * NAB: THIS SEEMS VERY INEFFICIENT */
00496     for (iatom=0; iatom<cell->natoms; iatom++) {
00497         atom = cell->atoms[iatom];
00498         atomID = atom->id;
00499         thee->atomFlags[atomID] = 0;
00500     }
00501
00502     return splineAcc(thee, center, win, infrad, cell);
00503 }
00504
00505 VPUBLIC void Vacc_splineAccGrad(Vacc *thee, double center[VAPBS_DIM],
00506           double win, double infrad, double *grad) {
00507
00508     int iatom, i, atomID;
00509     double acc = 1.0;
00510     double tgrad[VAPBS_DIM];
00511     VclistCell *cell;
00512     Vatom *atom = VNULL;
00513
00514     VASSERT(thee != NULL);
00515
00516     if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00517         Vnm_print(2, "Vacc_splineAccGrad: Vclist max_radius=%g;\n",
00518                   Vclist_maxRadius(thee->clist));
00519         Vnm_print(2, "Vacc_splineAccGrad: Insufficient for win=%g, infrad=%g\n",
00520                   win, infrad);
00521         VASSERT(0);
00522     }
00523
00524     /* Reset the gradient */
00525     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00526
00527     /* Get the cell; check for nullity */
00528     cell = Vclist_getCell(thee->clist, center);
00529     if (cell == VNULL) return;
00530
00531     /* Reset the list of atom flags */
00532     for (iatom=0; iatom<cell->natoms; iatom++) {
00533         atom = cell->atoms[iatom];
00534         atomID = atom->id;
00535         thee->atomFlags[atomID] = 0;
00536     }
00537
00538     /* Get the local accessibility */
00539     acc = splineAcc(thee, center, win, infrad, cell);

```

```

00540
00541     /* Accumulate the gradient of all local atoms */
00542     if (acc > VSMALL) {
00543         for (iatom=0; iatom<cell->natoms; iatom++) {
00544             atom = cell->atoms[iatom];
00545             Vacc_splineAccGradAtomNorm(thee, center, win, infrad, atom, tgrad);
00546         }
00547         for (i=0; i<VAPBS_DIM; i++) grad[i] += tgrad[i];
00548     }
00549     for (i=0; i<VAPBS_DIM; i++) grad[i] *= -acc;
00550 }
00551
00552 VPUBLIC double Vacc_molAcc(Vacc *thee, double center[VAPBS_DIM],
00553                     double radius) {
00554
00555     double rc;
00556
00557     /* ***** CHECK IF OUTSIDE ATOM+PROBE RADIUS SURFACE **** */
00558     if (Vacc_ivdwAcc(thee, center, radius) == 1.0) {
00559
00560         /* Vnm_print(2, "DEBUG: ivdwAcc = 1.0\n"); */
00561         rc = 1.0;
00562
00563         /* ***** CHECK IF INSIDE ATOM RADIUS SURFACE **** */
00564     } else if (Vacc_vdwAcc(thee, center) == 0.0) {
00565
00566         /* Vnm_print(2, "DEBUG: vdwAcc = 0.0\n"); */
00567         rc = 0.0;
00568
00569         /* ***** CHECK IF OUTSIDE MOLECULAR SURFACE **** */
00570     } else {
00571
00572         /* Vnm_print(2, "DEBUG: calling fastMolAcc...\n"); */
00573         rc = Vacc_fastMolAcc(thee, center, radius);
00574
00575     }
00576
00577     return rc;
00578
00579 }
00580
00581 VPUBLIC double Vacc_fastMolAcc(Vacc *thee, double center[VAPBS_DIM],
00582                     double radius) {
00583
00584     Vatom *atom;
00585     VaccSurf *surf;
00586     VclistCell *cell;
00587     int ipt, iatom, atomID;
00588     double dist2, rad2;
00589
00590     rad2 = radius*radius;
00591
00592     /* Check to see if the SAS has been defined */
00593     if (thee->srf == VNNULL) Vacc_SASA(thee, radius);
00594
00595     /* Get the cell associated with this point */
00596     cell = Vclist_getCell(thee->clist, center);

```

```

00597     if (cell == VNULL) {
00598         Vnm_print(2, "Vacc_fastMolAcc: unexpected VNULL VclistCell!\n");
00599         return 1.0;
00600     }
00601
00602     /* Loop through all the atoms in the cell */
00603     for (iatom=0; iatom<cell->natoms; iatom++) {
00604         atom = cell->atoms[iatom];
00605         atomID = Vatom_getAtomID(atom);
00606         surf = thee->surf[atomID];
00607         /* Loop through all SAS points associated with this atom */
00608         for (ipt=0; ipt<surf->npts; ipt++) {
00609             /* See if we're within a probe radius of the point */
00610             dist2 = VSQR(center[0]-(surf->xpts[ipt]))
00611                 + VSQR(center[1]-(surf->ypts[ipt]))
00612                 + VSQR(center[2]-(surf->zpts[ipt]));
00613             if (dist2 < rad2) return 1.0;
00614         }
00615     }
00616
00617     /* If all else failed, we are not inside the molecular surface */
00618     return 0.0;
00619 }
00620
00621
00622 #if defined(HAVE_MC_H)
00623 VPUBLIC void Vacc_writeGMV(Vacc *thee, double radius, int meth, Gem *gm,
00624     char *iodev, char *iofmt, char *iohost, char *iofile) {
00625
00626     double *accVals[MAXV], coord[3];
00627     Vio *sock;
00628     int invert, icoord;
00629
00630     for (invert=0; invert<MAXV; invert++) accVals[invert] = VNULL;
00631     accVals[0] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double));
00632     accVals[1] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double));
00633     for (invert=0; invert<Gem_numVV(gm); invert++) {
00634         for (icoord=0; icoord<3; icoord++)
00635             coord[icoord] = VV_coord(Gem_VV(gm, invert), icoord);
00636         if (meth == 0) {
00637             accVals[0][invert] = Vacc_molAcc(thee, coord, radius);
00638             accVals[1][invert] = Vacc_molAcc(thee, coord, radius);
00639         } else if (meth == 1) {
00640             accVals[0][invert] = Vacc_ivdwAcc(thee, coord, radius);
00641             accVals[1][invert] = Vacc_ivdwAcc(thee, coord, radius);
00642         } else if (meth == 2) {
00643             accVals[0][invert] = Vacc_vdwAcc(thee, coord);
00644             accVals[1][invert] = Vacc_vdwAcc(thee, coord);
00645         } else VASSERT(0);
00646     }
00647     sock = Vio_ctor(iodev, iofmt, iohost, iofile, "w");
00648     Gem_writeGMV(gm, sock, 1, accVals);
00649     Vio_dtors(&sock);
00650     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00651             (void **)&(accVals[0]));
00652     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00653             (void **)&(accVals[1]));

```

```

00654 }
00655 #endif /* defined(HAVE_MC_H) */
00656
00657 VPUBLIC double Vacc_SASA(Vacc *thee, double radius) {
00658     int i, natom;
00659     double area, *apos;
00660     Vatom *atom;
00661     VaccSurf *asurf;
00662
00663     unsigned long long mbeg;
00664
00665     natom = Valist_getNumberAtoms(thee->alist);
00666
00667     /* Check to see if we need to build the surface */
00668     if (thee->surf == NULL) {
00669         thee->surf = Vmem_malloc(thee->mem, natom, sizeof(VaccSurf *));
00670
00671 #if defined(DEBUG_MAC OSX_OCL) || defined(DEBUG_MAC OSX_STANDARD)
00672 #include "mach_chud.h"
00673     machm_(&mbeg);
00674 #pragma omp parallel for private(i,atom)
00675 #endif
00676     for (i=0; i<natom; i++) {
00677         atom = Valist_getAtom(thee->alist, i);
00678         /* NOTE: RIGHT NOW WE DO THIS FOR THE ENTIRE MOLECULE WHICH IS
00679          * INCREDIBLY INEFFICIENT, PARTICULARLY DURING FOCUSING!!! */
00680         thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere,
00681                                         radius);
00682     }
00683 }
00684
00685     /* Calculate the area */
00686     area = 0.0;
00687     for (i=0; i<natom; i++) {
00688         atom = Valist_getAtom(thee->alist, i);
00689         asurf = thee->surf[i];
00690         /* See if this surface needs to be rebuilt */
00691         if (asurf->probe_radius != radius) {
00692             Vnm_print(2, "Vacc_SASA: Warning -- probe radius changed from %g to
00693 %g!\n",
00694             asurf->probe_radius, radius);
00695             VaccSurf_dtor2(asurf);
00696             thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
00697             asurf = thee->surf[i];
00698         }
00699         area += (asurf->area);
00700     }
00701
00702 #if defined(DEBUG_MAC OSX_OCL) || defined(DEBUG_MAC OSX_STANDARD)
00703     mets_(&mbeg, "Vacc_SASA - Parallel");
00704 #endif
00705
00706     return area;
00707
00708 }
00709

```

```

00710 VPUBLIC double Vacc_totalSASA(Vacc *thee, double radius) {
00711     return Vacc_SASA(thee, radius);
00713
00714 }
00715
00716 VPUBLIC double Vacc_atomSASA(Vacc *thee, double radius, Vatom *atom) {
00717
00718     VaccSurf *asurf;
00719     int id;
00720
00721     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00722
00723     id = Vatom_getAtomID(atom);
00724     asurf = thee->surf[id];
00725
00726     /* See if this surface needs to be rebuilt */
00727     if (asurf->probe_radius != radius) {
00728         Vnm_print(2, "Vacc_SASA: Warning -- probe radius changed from %g to %g!\n",
00729                 asurf->probe_radius, radius);
00730         VaccSurf_dtor2(asurf);
00731         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
00732         asurf = thee->surf[id];
00733     }
00734
00735     return asurf->area;
00736
00737 }
00738
00739 VPUBLIC VaccSurf* VaccSurf_ctor(Vmem *mem, double probe_radius, int nsphere) {
00740     VaccSurf *thee;
00741
00742     //thee = Vmem_malloc(mem, 1, sizeof(Vacc));
00743     if (nsphere >= MAX_SPHERE PTS) {
00744         Vnm_print(2, "VaccSurf_ctor: Error! The requested number of grid points (%d)
00745 exceeds the maximum (%d)!%n", nsphere, MAX_SPHERE PTS);
00746         Vnm_print(2, "VaccSurf_ctor: Please check the variable MAX_SPHERE PTS to reset
00747 .%n");
00748         VASSERT(0);
00749     }
00750     thee = (VaccSurf*)calloc(1,sizeof(Vacc));
00751     VASSERT( VaccSurf_ctor2(thee, mem, probe_radius, nsphere) );
00752
00753     return thee;
00754 }
00755
00756 VPUBLIC int VaccSurf_ctor2(VaccSurf *thee, Vmem *mem, double probe_radius,
00757                               int nsphere) {
00758
00759     if (thee == VNULL) return 0;
00760
00761     thee->mem = mem;
00762     thee->npts = nsphere;
00763     thee->probe_radius = probe_radius;
00764     thee->area = 0.0;
00765

```

```

00764     if (thee->npts > 0) {
00765     /*
00766         thee->xpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00767         thee->ypts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00768         thee->zpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00769         thee->bpts = Vmem_malloc(thee->mem, thee->npts, sizeof(char));
00770     */
00771     thee->xpts = (double*)calloc(thee->npts, sizeof(double));
00772     thee->ypts = (double*)calloc(thee->npts, sizeof(double));
00773     thee->zpts = (double*)calloc(thee->npts, sizeof(double));
00774     thee->bpts = (char*)calloc(thee->npts, sizeof(char));
00775 } else {
00776     thee->xpts = VNULL;
00777     thee->ypts = VNULL;
00778     thee->zpts = VNULL;
00779     thee->bpts = VNULL;
00780 }
00781
00782     return 1;
00783 }
00784
00785 VPUBLIC void VaccSurf_dtor(VaccSurf **thee) {
00786     Vmem *mem;
00787
00788     if ((*thee) != VNULL) {
00789         mem = (*thee)->mem;
00790         VaccSurf_dtor2(*thee);
00791         //Vmem_free(mem, 1, sizeof(VaccSurf), (void **)thee);
00792         free(*thee);
00793         (*thee) = VNULL;
00794     }
00795 }
00796
00797 }
00798
00799 VPUBLIC void VaccSurf_dtor2(VaccSurf *thee) {
00800
00801     if (thee->npts > 0) {
00802     /*
00803         Vmem_free(thee->mem, thee->npts, sizeof(double),
00804             (void **)&(thee->xpts));
00805         Vmem_free(thee->mem, thee->npts, sizeof(double),
00806             (void **)&(thee->ypts));
00807         Vmem_free(thee->mem, thee->npts, sizeof(double),
00808             (void **)&(thee->zpts));
00809         Vmem_free(thee->mem, thee->npts, sizeof(char),
00810             (void **)&(thee->bpts));
00811     */
00812     free(thee->xpts);
00813     free(thee->ypts);
00814     free(thee->zpts);
00815     free(thee->bpts);
00816 }
00817 }
00818
00819 VPUBLIC VaccSurf* Vacc_atomSurf(Vacc *thee, Vatom *atom,
00820                                     VaccSurf *ref, double prad) {

```

```

00821
00822     VaccSurf *surf;
00823     int i, j, npts, atomID;
00824     double arad, rad, pos[3], *apos;
00825     char bpts[MAX_SPHERE PTS];
00826
00827     /* Get atom information */
00828     arad = Vatom_getRadius(atom);
00829     apos = Vatom_getPosition(atom);
00830     atomID = Vatom_getAtomID(atom);
00831
00832     if (arad < VSMALL) {
00833         return VaccSurf_ctor(thee->mem, prad, 0);
00834     }
00835
00836     rad = arad + prad;
00837
00838     /* Determine which points will contribute */
00839     npts = 0;
00840     for (i=0; i<ref->npts; i++) {
00841         /* Reset point flag: zero-radius atoms do not contribute */
00842         pos[0] = rad*(ref->xpts[i]) + apos[0];
00843         pos[1] = rad*(ref->ypts[i]) + apos[1];
00844         pos[2] = rad*(ref->zpts[i]) + apos[2];
00845         if (ivdwAccExclus(thee, pos, prad, atomID)) {
00846             npts++;
00847             bpts[i] = 1;
00848         } else {
00849             bpts[i] = 0;
00850         }
00851     }
00852
00853     /* Allocate space for the points */
00854     surf = VaccSurf_ctor(thee->mem, prad, npts);
00855
00856     /* Assign the points */
00857     j = 0;
00858     for (i=0; i<ref->npts; i++) {
00859         if (bpts[i]) {
00860             surf->bpts[j] = 1;
00861             surf->xpts[j] = rad*(ref->xpts[i]) + apos[0];
00862             surf->ypts[j] = rad*(ref->ypts[i]) + apos[1];
00863             surf->zpts[j] = rad*(ref->zpts[i]) + apos[2];
00864             j++;
00865         }
00866     }
00867
00868     /* Assign the area */
00869     surf->area = 4.0*VPI*rad*rad*((double)(surf->npts))/((double)(ref->npts));
00870
00871     return surf;
00872
00873 }
00874
00875 VPUBLIC VaccSurf* VaccSurf_refSphere(Vmem *mem, int npts) {
00876     VaccSurf *surf;

```

```

00878     int nactual, i, itheta, ntheta, iphi, nphimax, nphi;
00879     double frac;
00880     double sintheta, costheta, theta, dtheta;
00881     double sinphi, cosphi, phi, dphi;
00882
00883     /* Setup "constants" */
00884     frac = ((double)(npts))/4.0;
00885     ntheta = VRINT(VSQRT(Vunit_pi*frac));
00886     dtheta = Vunit_pi/((double)(ntheta));
00887     nphimax = 2*ntheta;
00888
00889     /* Count the actual number of points to be used */
00890     nactual = 0;
00891     for (itheta=0; itheta<ntheta; itheta++) {
00892         theta = dtheta*((double)(itheta));
00893         sintheta = VSIN(theta);
00894         costheta = VCOS(theta);
00895         nphi = VRINT(sintheta*nphimax);
00896         nactual += nphi;
00897     }
00898
00899     /* Allocate space for the points */
00900     surf = VaccSurf_ctor(mem, 1.0, nactual);
00901
00902     /* Clear out the boolean array */
00903     for (i=0; i<nactual; i++) surf->bpts[i] = 1;
00904
00905     /* Assign the points */
00906     nactual = 0;
00907     for (itheta=0; itheta<ntheta; itheta++) {
00908         theta = dtheta*((double)(itheta));
00909         sintheta = VSIN(theta);
00910         costheta = VCOS(theta);
00911         nphi = VRINT(sintheta*nphimax);
00912         if (nphi != 0) {
00913             dphi = 2*Vunit_pi/((double)(nphi));
00914             for (iphi=0; iphi<nphi; iphi++) {
00915                 phi = dphi*((double)(iphi));
00916                 sinphi = VSIN(phi);
00917                 cosphi = VCOS(phi);
00918                 surf->xpts[nactual] = cosphi * sintheta;
00919                 surf->ypts[nactual] = sinphi * sintheta;
00920                 surf->zpts[nactual] = costheta;
00921                 nactual++;
00922             }
00923         }
00924     }
00925
00926     surf->npts = nactual;
00927
00928     return surf;
00929 }
00930
00931 VPUBLIC VaccSurf* Vacc_atomSASPoints(Vacc *thee, double radius,
00932                                         Vatom *atom) {
00933
00934     VaccSurf *asurf = VNULL;

```

```

00935     int id;
00936
00937     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00938     id = Vatom_getAtomID(atom);
00939
00940     asurf = thee->surf[id];
00941
00942     /* See if this surface needs to be rebuilt */
00943     if (asurf->probe_radius != radius) {
00944         Vnm_print(2, "Vacc_SASA: Warning -- probe radius changed from %g to %g!\n",
00945         n",
00946         asurf->probe_radius, radius);
00947         VaccSurf_dtor2(asurf);
00948         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
00949     }
00950
00951     return asurf;
00952
00953 }
00954
00955 VPUBLIC void Vacc_splineAccGradAtomNorm4(Vacc *thee, double center[VAPBS_DIM],
00956             double win, double infrad, Vatom *atom, double *grad) {
00957
00958     int i;
00959     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5, sm6, sm7;
00960     double e, e2, e3, e4, e5, e6, e7;
00961     double b, b2, b3, b4, b5, b6, b7;
00962     double c0, c1, c2, c3, c4, c5, c6, c7;
00963     double denom, mygrad;
00964     double mychi = 1.0;           /* Char. func. value for given atom */
00965
00966     VASSERT(thee != NULL);
00967
00968     /* The grad is zero by default */
00969     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00970
00971     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00972      * *** MAGNITUDE OF THE FORCE *** */
00973     apos = Vatom_getPosition(atom);
00974     /* Zero-radius atoms don't contribute */
00975     if (Vatom_getRadius(atom) > 0.0) {
00976
00977         arad = Vatom_getRadius(atom);
00978         arad = arad + infrad;
00979         b = arad - win;
00980         e = arad + win;
00981
00982         e2 = e * e;
00983         e3 = e2 * e;
00984         e4 = e3 * e;
00985         e5 = e4 * e;
00986         e6 = e5 * e;
00987         e7 = e6 * e;
00988         b2 = b * b;
00989         b3 = b2 * b;
00990         b4 = b3 * b;

```

```

00991     b5 = b4 * b;
00992     b6 = b5 * b;
00993     b7 = b6 * b;
00994
00995     denom = e7 - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
00996     + 35.0*e3*b4 - 21.0*b5*e2 + 7.0*e*b6 - b7;
00997     c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
00998     c1 = -140.0*b3*e3/denom;
00999     c2 = 210.0*e2*b2*(e + b)/denom;
01000     c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
01001     c4 = 35.0*(e3 + 9.0*b*e2 + 9.0*e*b2 + b3)/denom;
01002     c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
01003     c6 = 70.0*(e + b)/denom;
01004     c7 = -20.0/denom;
01005
01006     dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
01007     + VSQR(apos[2]-center[2]));
01008
01009     /* If we're inside an atom, the entire characteristic function
01010 * will be zero and the grad will be zero, so we can stop */
01011     if (dist < (arad - win)) return;
01012     /* Likewise, if we're outside the smoothing window, the characteristic
01013 * function is unity and the grad will be zero, so we can stop */
01014     else if (dist > (arad + win)) return;
01015     /* Account for floating point error at the border
01016 * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
01017 * (Vacc_SplineAccAtom)? */
01018     else if ((VABS(dist - (arad - win)) < VSMALL) ||
01019               (VABS(dist - (arad + win)) < VSMALL)) return;
01020     /* If we're inside the smoothing window */
01021     else {
01022         sm = dist;
01023         sm2 = sm * sm;
01024         sm3 = sm2 * sm;
01025         sm4 = sm3 * sm;
01026         sm5 = sm4 * sm;
01027         sm6 = sm5 * sm;
01028         sm7 = sm6 * sm;
01029         mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01030         + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
01031         mygrad = c1 + 2.0*c2*sm + 3.0*c3*sm2 + 4.0*c4*sm3
01032         + 5.0*c5*sm4 + 6.0*c6*sm5 + 7.0*c7*sm6;
01033         if (mychi <= 0.0) {
01034             /* Avoid numerical round off errors */
01035             return;
01036             } else if (mychi > 1.0) {
01037             /* Avoid numerical round off errors */
01038             mychi = 1.0;
01039             }
01040             }
01041             /* Now assemble the grad vector */
01042             VASSERT(mychi > 0.0);
01043             for (i=0; i<VAPBS_DIM; i++)
01044                 grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
01045             }
01046     }
01047

```

```

01048 VPUBLIC void Vacc_splineAccGradAtomNorm3(Vacc *thee, double center[VAPBS_DIM],
01049         double win, double infrad, Vatom *atom, double *grad) {
01050
01051     int i;
01052     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5;
01053     double e, e2, e3, e4, e5;
01054     double b, b2, b3, b4, b5;
01055     double c0, c1, c2, c3, c4, c5;
01056     double denom, mygrad;
01057     double mychi = 1.0;           /* Char. func. value for given atom */
01058
01059     VASSERT(thee != NULL);
01060
01061     /* The grad is zero by default */
01062     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
01063
01064     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
01065     * *** MAGNITUDE OF THE FORCE *** */
01066     apos = Vatom_getPosition(atom);
01067     /* Zero-radius atoms don't contribute */
01068     if (Vatom_getRadius(atom) > 0.0) {
01069
01070         arad = Vatom_getRadius(atom);
01071         arad = arad + infrad;
01072         b = arad - win;
01073         e = arad + win;
01074
01075         e2 = e * e;
01076         e3 = e2 * e;
01077         e4 = e3 * e;
01078         e5 = e4 * e;
01079         b2 = b * b;
01080         b3 = b2 * b;
01081         b4 = b3 * b;
01082         b5 = b4 * b;
01083
01084         denom = pow((e - b), 5.0);
01085         c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
01086         c1 = 30.0*e2*b2;
01087         c2 = -30.0*(e2*b + e*b2);
01088         c3 = 10.0*(e2 + 4.0*e*b + b2);
01089         c4 = -15.0*(e + b);
01090         c5 = 6;
01091         c0 = c0/denom;
01092         c1 = c1/denom;
01093         c2 = c2/denom;
01094         c3 = c3/denom;
01095         c4 = c4/denom;
01096         c5 = c5/denom;
01097
01098         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
01099             + VSQR(apos[2]-center[2]));
01100
01101         /* If we're inside an atom, the entire characteristic function
01102         * will be zero and the grad will be zero, so we can stop */
01103         if (dist < (arad - win)) return;
01104         /* Likewise, if we're outside the smoothing window, the characteristic

```

```

01105     * function is unity and the grad will be zero, so we can stop */
01106     else if (dist > (arad + win)) return;
01107     /* Account for floating point error at the border
01108 * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
01109 * (Vacc_SplineAccAtom)? */
01110     else if ((VABS(dist - (arad - win)) < VSMALL) ||
01111         (VABS(dist - (arad + win)) < VSMALL)) return;
01112     /* If we're inside the smoothing window */
01113     else {
01114         sm = dist;
01115         sm2 = sm * sm;
01116         sm3 = sm2 * sm;
01117         sm4 = sm3 * sm;
01118         sm5 = sm4 * sm;
01119         mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01120         + c4*sm4 + c5*sm5;
01121         mygrad = c1 + 2.0*c2*sm + 3.0*c3*sm2 + 4.0*c4*sm3
01122         + 5.0*c5*sm4;
01123         if (mychi <= 0.0) {
01124             /* Avoid numerical round off errors */
01125             return;
01126         } else if (mychi > 1.0) {
01127             /* Avoid numerical round off errors */
01128             mychi = 1.0;
01129         }
01130     }
01131     /* Now assemble the grad vector */
01132     VASSERT(mychi > 0.0);
01133     for (i=0; i<VAPBS_DIM; i++)
01134         grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
01135     }
01136 }
01137
01138 /* //////////////////////////////// Routine: Vacc_atomdSAV
01139 // Routine: Vacc_atomdSAV
01140 //
01141 // Purpose: Calculates the vector valued atomic derivative of volume
01142 //
01143 // Args:      radius  The radius of the solvent probe in Angstroms
01144 //           iatom    Index of the atom in thee->alist
01145 //
01146 // Author:    Jason Wagoner
01147 //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01148 VPUBLIC void Vacc_atomdSAV(Vacc *thee, double srad, Vatom *atom, double *dSA) {
01149
01150     int ipt, iatom;
01151
01152     double area;
01153     double *tPos, tRad, vec[3];
01154     double dx,dy,dz;
01155     VaccSurf *ref;
01156     dx = 0.0;
01157     dy = 0.0;
01158     dz = 0.0;
01159     /* Get the atom information */
01160     ref = thee->refSphere;
01161     iatom = Vatom_getAtomID(atom);
01162

```

```

01163
01164     dSA[0] = 0.0;
01165     dSA[1] = 0.0;
01166     dSA[2] = 0.0;
01167
01168     tPos = Vatom_getPosition(atom);
01169     tRad = Vatom_getRadius(atom);
01170
01171     if(tRad == 0.0) return;
01172
01173     area = 4.0*VPI*(tRad+srad)*(tRad+srad)/((double)(ref->npts));
01174     for (ipt=0; ipt<ref->npts; ipt++) {
01175         vec[0] = (tRad+srad)*ref->xpts[ipt] + tPos[0];
01176         vec[1] = (tRad+srad)*ref->ypts[ipt] + tPos[1];
01177         vec[2] = (tRad+srad)*ref->zpts[ipt] + tPos[2];
01178         if (ivdwAccExclus(thee, vec, srad, iatom)) {
01179             dx = dx+vec[0]-tPos[0];
01180             dy = dy+vec[1]-tPos[1];
01181             dz = dz+vec[2]-tPos[2];
01182         }
01183     }
01184
01185     if ((tRad+srad) != 0){
01186         dSA[0] = dx*area/(tRad+srad);
01187         dSA[1] = dy*area/(tRad+srad);
01188         dSA[2] = dz*area/(tRad+srad);
01189     }
01190
01191 }
01192
01193 /* Note: This is purely test code to make certain that the dSASA code is
01194    behaving properly. This function should NEVER be called by anyone
01195    other than an APBS developer at Wash U.
01196 */
01197 VPRIVATE double Vacc_SASAPos(Vacc *thee, double radius) {
01198
01199     int i, natom;
01200     double area;
01201     Vatom *atom;
01202     VaccSurf *asurf;
01203
01204     natom = Valist_getNumberAtoms(thee->alist);
01205
01206     /* Calculate the area */
01207     area = 0.0;
01208     for (i=0; i<natom; i++) {
01209         atom = Valist_getAtom(thee->alist, i);
01210         asurf = thee->surf[i];
01211
01212         VaccSurf_dtor2(asurf);
01213         thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
01214         asurf = thee->surf[i];
01215         area += (asurf->area);
01216     }
01217
01218     return area;
01219

```

```

01220 }
01221
01222 VPRIIVATE double Vacc_atomSASAPos(Vacc *thee, double radius, Vatom *atom,int mode)
01223 {
01224     VaccSurf *asurf;
01225     int id;
01226     static int warned = 0;
01227
01228     if ((thee->surf == VNULL) || (mode == 1)){
01229         if(!warned){
01230             printf("WARNING: Recalculating entire surface!!!!\n");
01231             warned = 1;
01232         }
01233         Vacc_SASAPos(thee, radius);
01234     }
01235
01236     id = Vatom_getAtomID(atom);
01237     asurf = thee->surf[id];
01238
01239     VaccSurf_dtor(&asurf);
01240     thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
01241     asurf = thee->surf[id];
01242
01243     return asurf->area;
01244
01245 }
01246
01247 /* /////////////////////////////////
01248 // Routine: Vacc_atomdSASA
01249 //
01250 // Purpose: Calculates the derivative of surface area with respect to atomic
01251 // displacement using finite difference methods.
01252 //
01253 // Args:      radius  The radius of the solvent probe in Angstroms
01254 //           iatom    Index of the atom in thee->alist
01255 //
01256 // Author:    Jason Wagoner
01257 //           David Gohara
01258 //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01259 VPUBLIC void Vacc_atomdSASA(Vacc *thee, double dpos, double srad, Vatom *atom, do
01260 uble *dSA) {
01261
01262     int iatom;
01263     double *temp_Pos, tRad;
01264     double tPos[3];
01265     double axb1,axt1,ayb1,ayt1,azb1,azt1;
01266     VaccSurf *ref;
01267
01268     /* Get the atom information */
01269     ref = thee->refSphere;
01270     temp_Pos = VatomGetPosition(atom);
01271     tRad = Vatom_getRadius(atom);
01272     iatom = Vatom_getAtomID(atom);
01273
01274     dSA[0] = 0.0;
01275     dSA[1] = 0.0;

```

```

01276     dSA[2] = 0.0;
01277
01278     tPos[0] = temp_Pos[0];
01279     tPos[1] = temp_Pos[1];
01280     tPos[2] = temp_Pos[2];
01281
01282     /* Shift by pos -/+ on x */
01283     temp_Pos[0] -= dpos;
01284     axb1 = Vacc_atomSASAPos(thee, srad, atom,0);
01285     temp_Pos[0] = tPos[0];
01286
01287     temp_Pos[0] += dpos;
01288     axt1 = Vacc_atomSASAPos(thee, srad, atom,0);
01289     temp_Pos[0] = tPos[0];
01290
01291     /* Shift by pos -/+ on y */
01292     temp_Pos[1] -= dpos;
01293     ayb1 = Vacc_atomSASAPos(thee, srad, atom,0);
01294     temp_Pos[1] = tPos[1];
01295
01296     temp_Pos[1] += dpos;
01297     ayt1 = Vacc_atomSASAPos(thee, srad, atom,0);
01298     temp_Pos[1] = tPos[1];
01299
01300     /* Shift by pos -/+ on z */
01301     temp_Pos[2] -= dpos;
01302     azb1 = Vacc_atomSASAPos(thee, srad, atom,0);
01303     temp_Pos[2] = tPos[2];
01304
01305     temp_Pos[2] += dpos;
01306     azt1 = Vacc_atomSASAPos(thee, srad, atom,0);
01307     temp_Pos[2] = tPos[2];
01308
01309     /* Reset the atom SASA to zero displacement */
01310     Vacc_atomSASAPos(thee, srad, atom,0);
01311
01312     /* Calculate the final value */
01313     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01314     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01315     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01316 }
01317
01318 /* Note: This is purely test code to make certain that the dSASA code is
01319    behaving properly. This function should NEVER be called by anyone
01320    other than an APBS developer at Wash U.
01321 */
01322 VPUBLIC void Vacc_totalAtomdSASA(Vacc *thee, double dpos, double srad, Vatom *ato
m, double *dSA) {
01323
01324     int iatom;
01325     double *temp_Pos, tRad;
01326     double tPos[3];
01327     double axb1, axt1, ayb1, ayt1, azb1, azt1;
01328     VaccSurf *ref;
01329
01330     /* Get the atom information */
01331     ref = thee->refSphere;

```

```

01332     temp_Pos = Vatom_getPosition(atom);
01333     tRad = Vatom_getRadius(atom);
01334     iatom = Vatom_getAtomID(atom);
01335
01336     dSA[0] = 0.0;
01337     dSA[1] = 0.0;
01338     dSA[2] = 0.0;
01339
01340     tPos[0] = temp_Pos[0];
01341     tPos[1] = temp_Pos[1];
01342     tPos[2] = temp_Pos[2];
01343
01344     /* Shift by pos -/+ on x */
01345     temp_Pos[0] -= dpos;
01346     axb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01347     temp_Pos[0] = tPos[0];
01348
01349     temp_Pos[0] += dpos;
01350     axt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01351     temp_Pos[0] = tPos[0];
01352
01353     /* Shift by pos -/+ on y */
01354     temp_Pos[1] -= dpos;
01355     ayb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01356     temp_Pos[1] = tPos[1];
01357
01358     temp_Pos[1] += dpos;
01359     ayt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01360     temp_Pos[1] = tPos[1];
01361
01362     /* Shift by pos -/+ on z */
01363     temp_Pos[2] -= dpos;
01364     azb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01365     temp_Pos[2] = tPos[2];
01366
01367     temp_Pos[2] += dpos;
01368     azt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01369     temp_Pos[2] = tPos[2];
01370
01371     /* Calculate the final value */
01372     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01373     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01374     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01375 }
01376
01377 /* Note: This is purely test code to make certain that the dSASA code is
01378    behaving properly. This function should NEVER be called by anyone
01379    other than an APBS developer at Wash U.
01380 */
01381 VPUBLIC void Vacc_totalAtomdSAV(Vacc *thee, double dpos, double srad, Vatom *atom
01382   , double *dSA, Vclist *clist) {
01383     int iatom;
01384     double *temp_Pos, tRad;
01385     double tPos[3];
01386     double axb1, axt1, ayb1, ayt1, azb1, azt1;
01387     VaccSurf *ref;

```

```

01388
01389     /* Get the atom information */
01390     ref = theee->refSphere;
01391     temp_Pos = Vatom_getPosition(atom);
01392     tRad = Vatom_getRadius(atom);
01393     iatom = Vatom_getAtomID(atom);
01394
01395     dSA[0] = 0.0;
01396     dSA[1] = 0.0;
01397     dSA[2] = 0.0;
01398
01399     tPos[0] = temp_Pos[0];
01400     tPos[1] = temp_Pos[1];
01401     tPos[2] = temp_Pos[2];
01402
01403     /* Shift by pos -/+ on x */
01404     temp_Pos[0] -= dpos;
01405     axb1 = Vacc_totalSAV(theee,clist, VNULL, srad);
01406     temp_Pos[0] = tPos[0];
01407
01408     temp_Pos[0] += dpos;
01409     axt1 = Vacc_totalSAV(theee,clist, VNULL, srad);
01410     temp_Pos[0] = tPos[0];
01411
01412     /* Shift by pos -/+ on y */
01413     temp_Pos[1] -= dpos;
01414     ayb1 = Vacc_totalSAV(theee,clist, VNULL, srad);
01415     temp_Pos[1] = tPos[1];
01416
01417     temp_Pos[1] += dpos;
01418     ayt1 = Vacc_totalSAV(theee,clist, VNULL, srad);
01419     temp_Pos[1] = tPos[1];
01420
01421     /* Shift by pos -/+ on z */
01422     temp_Pos[2] -= dpos;
01423     azb1 = Vacc_totalSAV(theee,clist, VNULL, srad);
01424     temp_Pos[2] = tPos[2];
01425
01426     temp_Pos[2] += dpos;
01427     azt1 = Vacc_totalSAV(theee,clist, VNULL, srad);
01428     temp_Pos[2] = tPos[2];
01429
01430     /* Calculate the final value */
01431     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01432     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01433     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01434 }
01435
01436 VPUBLIC double Vacc_totalSAV(Vacc *theee, Vclist *clist, APOLparm *apolparm, doubl
e radius) {
01437
01438     int i;
01439     int npts[3];
01440
01441     double spacs[3], vec[3];
01442     double w, wx, wy, wz, len, fn, x, y, z, vol;
01443     double vol_density,sav;

```

```

01444     double *lower_corner, *upper_corner;
01445
01446     sav = 0.0;
01447     vol = 1.0;
01448     vol_density = 2.0;
01449
01450     lower_corner = clist->lower_corner;
01451     upper_corner = clist->upper_corner;
01452
01453     for (i=0; i<3; i++) {
01454         len = upper_corner[i] - lower_corner[i];
01455         vol *= len;
01456         fn = len*vol_density + 1;
01457         npts[i] = (int)ceil(fn);
01458         spacs[i] = len/((double)(npts[i])-1.0);
01459         if (apolparm != VNULL) {
01460             if (apolparm->setgrid) {
01461                 if (apolparm->grid[i] > spacs[i]) {
01462                     Vnm_print(2, "Vacc_totalSAV: Warning, your GRID value (%g) is larger than t
he recommended value (%g)!\n",
01463                     apolparm->grid[i], spacs[i]);
01464                 }
01465                 spacs[i] = apolparm->grid[i];
01466             }
01467         }
01468     }
01469 }
01470
01471     for (x=lower_corner[0]; x<=upper_corner[0]; x=x+spacs[0]) {
01472         if ( VABS(x - lower_corner[0]) < VSMALL) {
01473             wx = 0.5;
01474         } else if ( VABS(x - upper_corner[0]) < VSMALL) {
01475             wx = 0.5;
01476         } else {
01477             wx = 1.0;
01478         }
01479         vec[0] = x;
01480         for (y=lower_corner[1]; y<=upper_corner[1]; y=y+spacs[1]) {
01481             if ( VABS(y - lower_corner[1]) < VSMALL) {
01482                 wy = 0.5;
01483             } else if ( VABS(y - upper_corner[1]) < VSMALL) {
01484                 wy = 0.5;
01485             } else {
01486                 wy = 1.0;
01487             }
01488             vec[1] = y;
01489             for (z=lower_corner[2]; z<=upper_corner[2]; z=z+spacs[2]) {
01490                 if ( VABS(z - lower_corner[2]) < VSMALL) {
01491                     wz = 0.5;
01492                 } else if ( VABS(z - upper_corner[2]) < VSMALL) {
01493                     wz = 0.5;
01494                 } else {
01495                     wz = 1.0;
01496                 }
01497                 vec[2] = z;
01498                 w = wx*wy*wz;

```

```

01500
01501     sav += (w*(1.0-Vacc_ivdwAcc(thee, vec, radius)));
01502
01503     } /* z loop */
01504     } /* y loop */
01505     } /* x loop */
01506
01507     w = spacs[0]*spacs[1]*spacs[2];
01508     sav *= w;
01509
01510     return sav;
01511 }
01512
01513 int Vacc_wcaEnergyAtom(Vacc *thee, APOLparm *apolparm, Valist *alist,
01514                           Vclist *clist, int iatom, double *value) {
01515
01516     int i;
01517     int npts[3];
01518     int pad = 14;
01519
01520     int xmin, ymin, zmin;
01521     int xmax, ymax, zmax;
01522
01523     double sigma6, sigma12;
01524
01525     double spacs[3], vec[3];
01526     double w, wx, wy, wz, len, fn, x, y, z, vol;
01527     double x2,y2,z2,r;
01528     double vol_density, energy, rho, srad;
01529     double psig, epsilon, watepsilon, sigma, watsigma, eni, chi;
01530
01531     double *pos;
01532     double *lower_corner, *upper_corner;
01533
01534     Vatom *atom = VNULL;
01535     VASSERT(apolparm != VNULL);
01536
01537     energy = 0.0;
01538     vol = 1.0;
01539     vol_density = 2.0;
01540
01541     lower_corner = clist->lower_corner;
01542     upper_corner = clist->upper_corner;
01543
01544     atom = Valist_getAtom(alist, iatom);
01545     pos = VatomGetPosition(atom);
01546
01547     /* Note: these are the original temporary water parameters... they have been
01548        replaced by entries in a parameter file:
01549     watsigma = 1.7683;
01550     watepsilon = 0.1521;
01551     watepsilon = watepsilon*4.184;
01552     */
01553
01554     srad = apolparm->srad;
01555     rho = apolparm->bconc;
01556     watsigma = apolparm->watsigma;

```

```

01557 watepsilon = apolparm->watepsilon;
01558 psig = atom->radius;
01559 epsilon = atom->epsilon;
01560 sigma = psig + watsigma;
01561 epsilon = VSQRT((epsilon * watepsilon));
01562
01563 /* parameters */
01564 sigma6 = VPOW(sigma, 6);
01565 sigma12 = VPOW(sigma, 12);
01566 /* OPLS-style radius: double sigmar = sigma*VPOW(2, (1.0/6.0)); */
01567
01568 xmin = pos[0] - pad;
01569 xmax = pos[0] + pad;
01570 ymin = pos[1] - pad;
01571 ymax = pos[1] + pad;
01572 zmin = pos[2] - pad;
01573 zmax = pos[2] + pad;
01574
01575 for (i=0; i<3; i++) {
01576     len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01577     vol *= len;
01578     fn = len*vol_density + 1;
01579     npts[i] = (int)ceil(fn);
01580     spacs[i] = 0.5;
01581     if (apolparm->setgrid) {
01582         if (apolparm->grid[i] > spacs[i]) {
01583             Vnm_print(2, "Vacc_totalSAV: Warning, your GRID value (%g) is larger than th
e recommended value (%g)!\n",
01584             apolparm->grid[i], spacs[i]);
01585         }
01586         spacs[i] = apolparm->grid[i];
01587     }
01588 }
01589
01590 for (x=xmin; x<=xmax; x=x+spacs[0]) {
01591     if ( VABS(x - xmin) < VSMALL) {
01592         wx = 0.5;
01593     } else if ( VABS(x - xmax) < VSMALL) {
01594         wx = 0.5;
01595     } else {
01596         wx = 1.0;
01597     }
01598     vec[0] = x;
01599     for (y=ymin; y<=ymax; y=y+spacs[1]) {
01600         if ( VABS(y - ymin) < VSMALL) {
01601             wy = 0.5;
01602         } else if ( VABS(y - ymax) < VSMALL) {
01603             wy = 0.5;
01604         } else {
01605             wy = 1.0;
01606         }
01607         vec[1] = y;
01608         for (z=zmin; z<=zmax; z=z+spacs[2]) {
01609             if ( VABS(z - zmin) < VSMALL) {
01610                 wz = 0.5;
01611             } else if ( VABS(z - zmax) < VSMALL) {
01612                 wz = 0.5;

```

```

01613     } else {
01614         wz = 1.0;
01615     }
01616     vec[2] = z;
01617
01618     w = wx*wy*wz;
01619
01620     chi = Vacc_ivdwAcc(thee, vec, srad);
01621
01622     if (VABS(chi) > VSMALL) {
01623
01624         x2 = VSQR(vec[0]-pos[0]);
01625         y2 = VSQR(vec[1]-pos[1]);
01626         z2 = VSQR(vec[2]-pos[2]);
01627         r = VSQRT(x2+y2+z2);
01628
01629         if (r <= 14 && r >= sigma) {
01630             eni = chi*rho*epsilon*(-2.0*sigma6/VPOW(r,6)+sigma12/VPOW(r,12));
01631         }else if (r <= 14){
01632             eni = -1.0*epsilon*chi*rho;
01633         }else{
01634             eni = 0.0;
01635         }
01636     }else{
01637         eni = 0.0;
01638     }
01639
01640     energy += eni*w;
01641
01642     } /* z loop */
01643     } /* y loop */
01644 } /* x loop */
01645
01646 w = spacs[0]*spacs[1]*spacs[2];
01647 energy *= w;
01648
01649 *value = energy;
01650
01651 return VRC_SUCCESS;
01652 }
01653
01654 VPUBLIC int Vacc_wcaEnergy(Vacc *acc, APOLparm *apolparm, Valist *alist,
01655                 Vclist *clist){
01656
01657     int iatom;
01658     int rc = 0;
01659
01660     double energy = 0.0;
01661     double tenergy = 0.0;
01662     double rho = apolparm->bconc;
01663
01664     /* Do a sanity check to make sure that watepsilon and watsigma are set
01665      * If not, return with an error. */
01666     if(apolparm->setwat == 0){
01667         Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilo
n.\n");
01668     return VRC_FAILURE;

```

```

01669 }
01670
01671 if (VABS(rho) < VSMALL) {
01672     apolparm->wcaEnergy = tenergy;
01673     return 1;
01674 }
01675
01676     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01677         rc = Vacc_wcaEnergyAtom(acc, apolparm, alist, clist, iatom, &energy);
01678         if(rc == 0) return 0;
01679
01680         tenergy += energy;
01681     }
01682
01683     apolparm->wcaEnergy = tenergy;
01684
01685     return VRC_SUCCESS;
01686
01687 }
01688
01689 VPUBLIC int Vacc_wcaForceAtom(Vacc *thee, APOLparm *apolparm, Vclist *clist,
01690             Vatom *atom, double *force){
01691     int i,si;
01692     int npts[3];
01693     int pad = 14;
01694
01695         int xmin, ymin, zmin;
01696         int xmax, ymax, zmax;
01697
01698         double sigma6, sigma12;
01699
01700     double spacs[3], vec[3], fpt[3];
01701         double w, wx, wy, wz, len, fn, x, y, z, vol;
01702     double x2,y2,z2,r;
01703     double vol_density, fo;
01704     double rho;
01705     double srad, psig, epsilon, watepsilon, sigma, watsigma, chi;
01706
01707     double *pos;
01708         double *lower_corner, *upper_corner;
01709
01710     VASSERT(apolparm != VNULL);
01711
01712     /* Do a sanity check to make sure that watepsilon and watsigma are set
01713      * If not, return with an error. */
01714     if(apolparm->setwat == 0){
01715         Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilon\n");
01716         return VRC_FAILURE;
01717     }
01718
01719     vol = 1.0;
01720     vol_density = 2.0;
01721
01722     lower_corner = clist->lower_corner;
01723     upper_corner = clist->upper_corner;
01724

```

```

01725 pos = Vatom_getPosition(atom);
01726
01727 /* Note: these are the temporary water parameters we used to use in this
01728 routine... they have been replaced by entries in a parameter file
01729 watsigma = 1.7683;
01730 watepsilon = 0.1521;
01731 watepsilon = watepsilon*4.184;
01732 */
01733 srad = apolparm->srad;
01734 rho = apolparm->bconc;
01735 watsigma = apolparm->watsigma;
01736 watepsilon = apolparm->watepsilon;
01737
01738 psig = atom->radius;
01739 epsilon = atom->epsilon;
01740 sigma = psig + watsigma;
01741 epsilon = VSQRT((epsilon * watepsilon));
01742
01743 /* parameters */
01744 sigma6 = VPOW(sigma,6);
01745 sigma12 = VPOW(sigma,12);
01746 /* OPLS-style radius: double sigmar = sigma*VPOW(2, (1.0/6.0)); */
01747
01748 for (i=0; i<3; i++) {
01749   len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01750   vol *= len;
01751   fn = len*vol_density + 1;
01752   npts[i] = (int)ceil(fn);
01753   spacs[i] = 0.5;
01754   force[i] = 0.0;
01755   if (apolparm->setgrid) {
01756     if (apolparm->grid[i] > spacs[i]) {
01757       Vnm_print(2, "Vacc_totalSAV: Warning, your GRID value (%g) is larger than th
e recommended value (%g)!\n",
01758               apolparm->grid[i], spacs[i]);
01759     }
01760     spacs[i] = apolparm->grid[i];
01761   }
01762 }
01763
01764 xmin = pos[0] - pad;
01765 xmax = pos[0] + pad;
01766 ymin = pos[1] - pad;
01767 ymax = pos[1] + pad;
01768 zmin = pos[2] - pad;
01769 zmax = pos[2] + pad;
01770
01771 for (x=xmin; x<=xmax; x=x+spacs[0]) {
01772   if ( VABS(x - xmin) < VSMALL) {
01773     wx = 0.5;
01774   } else if ( VABS(x - xmax) < VSMALL) {
01775     wx = 0.5;
01776   } else {
01777     wx = 1.0;
01778   }
01779   vec[0] = x;
01780   for (y=ymin; y<=ymax; y=y+spacs[1]) {

```

```

01781     if ( VABS(y - ymin) < VSMALL) {
01782         wy = 0.5;
01783     } else if ( VABS(y - ymax) < VSMALL) {
01784         wy = 0.5;
01785     } else {
01786         wy = 1.0;
01787     }
01788     vec[1] = y;
01789     for (z=zmin; z<=zmax; z=z+spacs[2]) {
01790         if ( VABS(z - zmin) < VSMALL) {
01791             wz = 0.5;
01792         } else if ( VABS(z - zmax) < VSMALL) {
01793             wz = 0.5;
01794         } else {
01795             wz = 1.0;
01796         }
01797         vec[2] = z;
01798
01799         w = wx*wy*wz;
01800
01801         chi = Vacc_ivdwAcc(thee, vec, srad);
01802
01803         if (chi != 0.0) {
01804
01805             x2 = VSQR(vec[0]-pos[0]);
01806             y2 = VSQR(vec[1]-pos[1]);
01807             z2 = VSQR(vec[2]-pos[2]);
01808             r = VSQRT(x2+y2+z2);
01809
01810             if (r <= 14 && r >= sigma){
01811
01812                 fo = 12.0*chi*rho*epsilon*(sigma6/VPOW(r,7)-sigma12/VPOW(r,13));
01813
01814                 fpt[0] = -1.0*(pos[0]-vec[0])*fo/r;
01815                 fpt[1] = -1.0*(pos[1]-vec[1])*fo/r;
01816                 fpt[2] = -1.0*(pos[2]-vec[2])*fo/r;
01817
01818             }else {
01819                 for (si=0; si < 3; si++) fpt[si] = 0.0;
01820             }
01821         }else {
01822             for (si=0; si < 3; si++) fpt[si] = 0.0;
01823         }
01824
01825         for(i=0;i<3;i++){
01826             force[i] += (w*fpt[i]);
01827         }
01828
01829     } /* z loop */
01830     } /* y loop */
01831 } /* x loop */
01832
01833 w = spacs[0]*spacs[1]*spacs[2];
01834 for(i=0;i<3;i++) force[i] *= w;
01835
01836 return VRC_SUCCESS;
01837 }
```

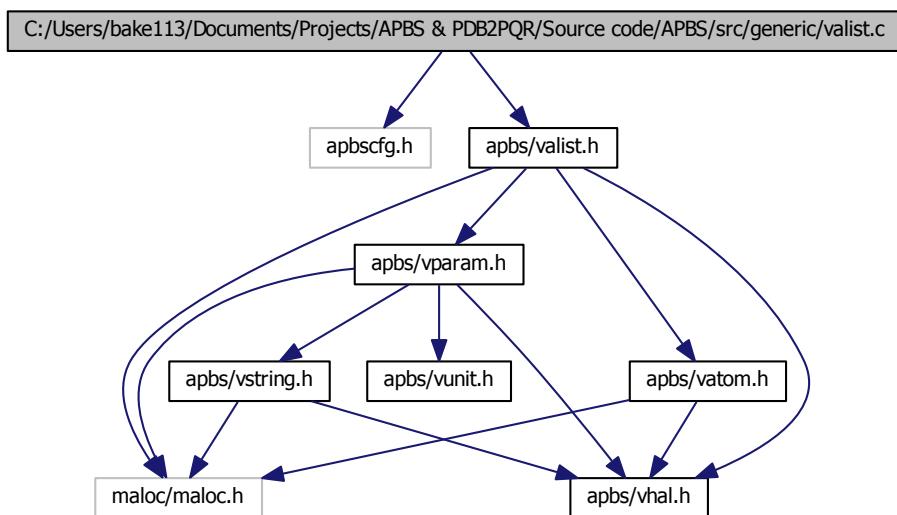
01838

10.63 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/valist.c File Reference

Class Valist methods.

```
#include "apbscfg.h"
#include "apbs/valist.h"

Include dependency graph for valist.c:
```



Functions

- VPUBLIC double [Valist_getCenterX](#) (*Valist *thee*)
Get x-coordinate of molecule center.
- VPUBLIC double [Valist_getCenterY](#) (*Valist *thee*)
Get y-coordinate of molecule center.
- VPUBLIC double [Valist_getCenterZ](#) (*Valist *thee*)

Get z-coordinate of molecule center.

- VPUBLIC `Vatom * Valist_getAtomList (Valist *thee)`
Get actual array of atom objects from the list.
- VPUBLIC int `Valist_getNumberAtoms (Valist *thee)`
Get number of atoms in the list.
- VPUBLIC `Vatom * Valist_getAtom (Valist *thee, int i)`
Get pointer to particular atom in list.
- VPUBLIC unsigned long int `Valist_memChk (Valist *thee)`
Get total memory allocated for this object and its members.
- VPUBLIC `Valist * Valist_ctor ()`
Construct the atom list object.
- VPUBLIC Vrc_Codes `Valist_ctor2 (Valist *thee)`
FORTRAN stub to construct the atom list object.
- VPUBLIC void `Valist_dtor (Valist **thee)`
Destroys atom list object.
- VPUBLIC void `Valist_dtor2 (Valist *thee)`
FORTRAN stub to destroy atom list object.
- VPRIVATE Vrc_Codes `Valist_readPDBSerial (Valist *thee, Vio *sock, int *serial)`

• VPRIVATE Vrc_Codes `Valist_readPDBAtomName (Valist *thee, Vio *sock, char atomName[VMAX_ARGLEN])`
• VPRIVATE Vrc_Codes `Valist_readPDBResidueName (Valist *thee, Vio *sock, char resName[VMAX_ARGLEN])`
• VPRIVATE Vrc_Codes `Valist_readPDBResidueNumber (Valist *thee, Vio *sock, int *resSeq)`
• VPRIVATE Vrc_Codes `Valist_readPDBAtomCoord (Valist *thee, Vio *sock, double *coord)`
• VPRIVATE Vrc_Codes `Valist_readPDBChargeRadius (Valist *thee, Vio *sock, double *charge, double *radius)`
• VPRIVATE Vrc_Codes `Valist_readPDB_throughXYZ (Valist *thee, Vio *sock, int *serial, char atomName[VMAX_ARGLEN], char resName[VMAX_ARGLEN], int *resSeq, double *x, double *y, double *z)`
• VPRIVATE `Vatom * Valist_getAtomStorage (Valist *thee, Vatom **plist, int *pnlist, int *pnatoms)`
• VPRIVATE Vrc_Codes `Valist_setAtomArray (Valist *thee, Vatom **plist, int nlist, int natoms)`
• VPUBLIC Vrc_Codes `Valist_readPDB (Valist *thee, Vparam *param, Vio *sock)`

Fill atom list with information from a PDB file.
- VPUBLIC Vrc_Codes `Valist_readPQR (Valist *thee, Vparam *params, Vio *sock)`

Fill atom list with information from a PQR file.

- VPUBLIC Vrc_Codes [Valist_readXML](#) (*Valist *thee, Vparam *params, Vio *sock*)

Fill atom list with information from an XML file.

- VPUBLIC Vrc_Codes [Valist_getStatistics](#) (*Valist *thee)*

Load up Valist with various statistics.

Variables

- VPRIVATE char * **Valist_whiteChars** = " \t\r\n"
- VPRIVATE char * **Valist_commChars** = "#%"
- VPRIVATE char * **Valist_xmlwhiteChars** = " \t\r\n<>"

10.63.1 Detailed Description

Class Valist methods.

Author

Nathan Baker

Version

Id:

[valist.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without

```

```
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [valist.c](#).

10.64 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/valist.c

```
00001  
00056 #include "apbscfg.h"  
00057 #include "apbs/valist.h"  
00058  
00059 VEMBED(rcsid="$Id: valist.c 1667 2011-12-02 23:22:02Z pcellis $" )  
00060  
00061 VPRIPRIVATE char *Valist_whiteChars = "\t\r\n";  
00062 VPRIPRIVATE char *Valist_commChars = "#%";  
00063 VPRIPRIVATE char *Valist_xmlwhiteChars = "\t\r\n<>";  
00064  
00065 #if !defined(VINLINE_VATOM)  
00066  
00067 VPUBLIC double Valist_getCenterX(Valist *thee) {  
00068  
00069     if (thee == NULL) {  
00070         Vnm_print(2, "Valist_getCenterX: Found null pointer when getting the center of  
X coordinate!\n");  
00071         VASSERT(0);  
00072     }  
00073     return thee->center[0];  
00074 }
```

```

00075 }
00076
00077 VPUBLIC double Valist_getCenterY(Valist *thee) {
00078
00079     if (thee == NULL) {
00080         Vnm_print(2, "Valist_getCenterY: Found null pointer when getting the center of
00081             Y coordinate!\n");
00082         VASSERT(0);
00083     }
00084     return thee->center[1];
00085 }
00086 VPUBLIC double Valist_getCenterZ(Valist *thee) {
00087
00088     if (thee == NULL) {
00089         Vnm_print(2, "Valist_getCenterZ: Found null pointer when getting the center of
00090             Z coordinate!\n");
00091         VASSERT(0);
00092     }
00093     return thee->center[2];
00094 }
00095
00096 VPUBLIC Vatom* Valist_getAtomList(Valist *thee) {
00097
00098     if (thee == NULL) {
00099         Vnm_print(2, "Valist_getAtomList: Found null pointer when getting the atom lis
00100             t!\\n");
00101         VASSERT(0);
00102     }
00103     return thee->atoms;
00104 }
00105
00106 VPUBLIC int Valist_getNumberAtoms(Valist *thee) {
00107
00108     if (thee == NULL) {
00109         Vnm_print(2, "Valist_getNumberAtoms: Found null pointer when getting the numbe
00110             r of atoms!\\n");
00111         VASSERT(0);
00112     }
00113     return thee->number;
00114 }
00115
00116 VPUBLIC Vatom* Valist_getAtom(Valist *thee, int i) {
00117
00118     if (thee == NULL) {
00119         Vnm_print(2, "Valist_getAtom: Found null pointer when getting atoms!\\n");
00120         VASSERT(0);
00121     }
00122     if (i >= thee->number) {
00123         Vnm_print(2, "Valist_getAtom: Requested atom number (%d) outside of atom list
00124             range (%d)!\\n", i, thee->number);
00125         VASSERT(0);
00126     }
00127     return &(thee->atoms[i]);

```

```
00127
00128 }
00129
00130 VPUBLIC unsigned long int Valist_memChk(Valist *thee) {
00131     if (thee == NULL) return 0;
00132     return Vmem_bytes(thee->vmem);
00133 }
00134
00135 }
00136
00137 #endif /* if !defined(VINLINE_VATOM) */
00138
00139 VPUBLIC Valist* Valist_ctor() {
00140
00141     /* Set up the structure */
00142     Valist *thee = VNULL;
00143     thee = Vmem_malloc(VNULL, 1, sizeof(Valist));
00144     if (thee == VNULL) {
00145         Vnm_print(2, "Valist_ctor: Got NULL pointer when constructing the atom list object!\n");
00146         VASSERT(0);
00147     }
00148     if (Valist_ctor2(thee) != VRC_SUCCESS) {
00149         Vnm_print(2, "Valist_ctor: Error in constructing the atom list object!\n");
00150         VASSERT(0);
00151     }
00152
00153     return thee;
00154 }
00155
00156 VPUBLIC Vrc_Codes Valist_ctor2(Valist *thee) {
00157
00158     thee->atoms = VNULL;
00159     thee->number = 0;
00160
00161     /* Initialize the memory management object */
00162     thee->vmem = Vmem_ctor("APBS:VALIST");
00163
00164     return VRC_SUCCESS;
00165 }
00166
00167
00168 VPUBLIC void Valist_dtor(Valist **thee)
00169 {
00170     if ((*thee) != VNULL) {
00171         Valist_dtor2(*thee);
00172         Vmem_free(VNULL, 1, sizeof(Valist), (void **)thee);
00173         (*thee) = VNULL;
00174     }
00175 }
00176
00177 VPUBLIC void Valist_dtor2(Valist *thee) {
00178
00179     Vmem_free(thee->vmem, thee->number, sizeof(Vatom), (void **)(&(thee->atoms)));
00180     thee->atoms = VNULL;
00181     thee->number = 0;
00182 }
```

```

00183     Vmem_dtor(&(thee->vmem));
00184 }
00185
00186 /* Read serial number from PDB ATOM/HETATM field */
00187 VPRIIVATE Vrc_Codes Valist_readPDBSerial(Valist *thee, Vio *sock, int *serial) {
00188
00189     char tok[VMAX_BUFSIZE];
00190     int ti = 0;
00191
00192     if (Vio_scanf(sock, "%s", tok) != 1) {
00193         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing serial!\n");
00194         return VRC_FAILURE;
00195     }
00196     if (sscanf(tok, "%d", &ti) != 1) {
00197         Vnm_print(2, "Valist_readPDB: Unable to parse serial token (%s) as int!\n",
00198                     tok);
00199         return VRC_FAILURE;
00200     }
00201     *serial = ti;
00202
00203     return VRC_SUCCESS;
00204 }
00205
00206 /* Read atom name from PDB ATOM/HETATM field */
00207 VPRIIVATE Vrc_Codes Valist_readPDBAtomName(Valist *thee, Vio *sock,
00208                                              char atomName[VMAX_ARGLEN]) {
00209
00210     char tok[VMAX_BUFSIZE];
00211
00212     if (Vio_scanf(sock, "%s", tok) != 1) {
00213         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing atom name!\n");
00214         return VRC_FAILURE;
00215     }
00216     if (strlen(tok) < VMAX_ARGLEN) strcpy(atomName, tok);
00217     else {
00218         Vnm_print(2, "Valist_readPDB: Atom name (%s) too long!\n", tok);
00219         return VRC_FAILURE;
00220     }
00221     return VRC_SUCCESS;
00222 }
00223
00224 /* Read residue name from PDB ATOM/HETATM field */
00225 VPRIIVATE Vrc_Codes Valist_readPDBResidueName(Valist *thee, Vio *sock,
00226                                              char resName[VMAX_ARGLEN]) {
00227
00228     char tok[VMAX_BUFSIZE];
00229
00230     if (Vio_scanf(sock, "%s", tok) != 1) {
00231         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing residue na-
00232 me!\n");
00233         return VRC_FAILURE;
00234     }
00235     if (strlen(tok) < VMAX_ARGLEN) strcpy(resName, tok);
00236     else {

```

```

00236         Vnm_print(2, "Valist_readPDB: Residue name (%s) too long!\n", tok);
00237         return VRC_FAILURE;
00238     }
00239     return VRC_SUCCESS;
00240 }
00241
00242 /* Read residue number from PDB ATOM/HETATM field */
00243 VPRIVATE Vrc_Codes Valist_readPDBResidueNumber(
00244     Valist *thee, Vio *sock, int *resSeq) {
00245
00246     char tok[VMAX_BUFSIZE];
00247     char *resstring;
00248     int ti = 0;
00249
00250     if (Vio_scanf(sock, "%s", tok) != 1) {
00251         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing resSeq!\n"
00252     );
00253         return VRC_FAILURE;
00254     }
00255     if (sscanf(tok, "%d", &ti) != 1) {
00256
00257         /* One of three things can happen here:
00258         1) There is a chainID in the line:    THR A   1
00259         2) The chainID is merged with resSeq: THR A1001
00260         3) An actual error:                  THR foo
00261
00262     */
00263     if (strlen(tok) == 1) {
00264
00265         /* Case 1: Chain ID Present
00266             Read the next field and hope its a float */
00267
00268         if (Vio_scanf(sock, "%s", tok) != 1) {
00269             Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing resSeq!\n"
00270         );
00271             return VRC_FAILURE;
00272         }
00273         if (sscanf(tok, "%d", &ti) != 1) {
00274             Vnm_print(2, "Valist_readPDB: Unable to parse resSeq token (%s) as int!\n",
00275                         tok);
00276             return VRC_FAILURE;
00277         }
00278     } else {
00279
00280         /* Case 2: Chain ID, merged string.
00281             Move pointer forward past the chainID and check
00282             */
00283         //strcpy(resstring, tok);
00284         resstring = tok;
00285         resstring++;
00286
00287         if (sscanf(resstring, "%d", &ti) != 1) {
00288             /* Case 3: More than one non-numeral char is present. Error.*/
00289             Vnm_print(2, "Valist_readPDB: Unable to parse resSeq token (%s)
00290             as int!\n",
00291                     resstring);
00292             return VRC_FAILURE;

```

```

00290     }
00291     }
00292     }
00293     *resSeq = ti;
00294     return VRC_SUCCESS;
00295 }
00296 }
00297
00298 /* Read atom coordinate from PDB ATOM/HETATOM field */
00299 VPRIVATE Vrc_Codes Valist_readPDBAtomCoord(Valist *thee, Vio *sock, double *coord
00300 ) {
00301     char tok[VMAX_BUFSIZE];
00302     double tf = 0;
00303
00304     if (Vio_scanf(sock, "%s", tok) != 1) {
00305         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing atom coordinate!\n");
00306         return VRC_FAILURE;
00307     }
00308     if (sscanf(tok, "%lf", &tf) != 1) {
00309         return VRC_FAILURE;
00310     }
00311     *coord = tf;
00312
00313     return VRC_SUCCESS;
00314 }
00315
00316 /* Read charge and radius from PQR ATOM/HETATOM field */
00317 VPRIVATE Vrc_Codes Valist_readPDBChargeRadius(Valist *thee, Vio *sock,
00318         double *charge, double *radius) {
00319
00320     char tok[VMAX_BUFSIZE];
00321     double tf = 0;
00322
00323     if (Vio_scanf(sock, "%s", tok) != 1) {
00324         Vnm_print(2, "Valist_readPQR: Ran out of tokens while parsing charge!\n");
00325         return VRC_FAILURE;
00326     }
00327     if (sscanf(tok, "%lf", &tf) != 1) {
00328         return VRC_FAILURE;
00329     }
00330     *charge = tf;
00331
00332     if (Vio_scanf(sock, "%s", tok) != 1) {
00333         Vnm_print(2, "Valist_readPQR: Ran out of tokens while parsing radius!\n");
00334         return VRC_FAILURE;
00335     }
00336     if (sscanf(tok, "%lf", &tf) != 1) {
00337         return VRC_FAILURE;
00338     }
00339     *radius = tf;
00340
00341     return VRC_SUCCESS;
00342 }

```

```

00343
00344 /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00345 VPRIVATE Vrc_Codes Valist_readPDB_throughXYZ(
00346     Valist *thee,
00347     Vio *sock, /* Socket ready for reading */
00348     int *serial, /* Set to atom number */
00349     char atomName[VMAX_ARGLEN], /* Set to atom name */
00350     char resName[VMAX_ARGLEN], /* Set to residue name */
00351     int *resSeq, /* Set to residue number */
00352     double *x, /* Set to x-coordinate */
00353     double *y, /* Set to y-coordinate */
00354     double *z /* Set to z-coordinate */
00355 ) {
00356
00357
00358     int i, njunk, gotit;
00359
00360     /* Grab serial */
00361     if (Valist_readPDBSerial(thee, sock, serial) == VRC_FAILURE) {
00362         Vnm_print(2, "Valist_readPDB: Error while parsing serial!\n");
00363     }
00364
00365     /* Grab atom name */
00366     if (Valist_readPDBAtomName(thee, sock, atomName) == VRC_FAILURE) {
00367         Vnm_print(2, "Valist_readPDB: Error while parsing atom name!\n");
00368         return VRC_FAILURE;
00369     }
00370
00371     /* Grab residue name */
00372     if (Valist_readPDBResidueName(thee, sock, resName) == VRC_FAILURE) {
00373         Vnm_print(2, "Valist_readPDB: Error while parsing residue name!\n");
00374         return VRC_FAILURE;
00375     }
00376
00377
00378     /* Grab residue number */
00379     if (Valist_readPDBResidueNumber(thee, sock, resSeq) == VRC_FAILURE) {
00380         Vnm_print(2, "Valist_readPDB: Error while parsing residue name!\n");
00381         return VRC_FAILURE;
00382     }
00383
00384
00385     /* Read tokens until we find one that can be parsed as an atom
00386      * x-coordinate. We will allow njunk=1 intervening field that
00387      * cannot be parsed as a coordinate */
00388     njunk = 1;
00389     gotit = 0;
00390     for (i=0; i<(njunk+1); i++) {
00391         if (Valist_readPDBAtomCoord(thee, sock, x) == VRC_SUCCESS) {
00392             gotit = 1;
00393             break;
00394         }
00395     }
00396     if (!gotit) {
00397         Vnm_print(2, "Valist_readPDB: Can't find x!\n");
00398         return VRC_FAILURE;
00399     }

```

```

00400     /* Read y-coordinate */
00401     if (Valist_readPDBAtomCoord(thee, sock, y) == VRC_FAILURE) {
00402         Vnm_print(2, "Valist_readPDB: Can't find y!\n");
00403         return VRC_FAILURE;
00404     }
00405     /* Read z-coordinate */
00406     if (Valist_readPDBAtomCoord(thee, sock, z) == VRC_FAILURE) {
00407         Vnm_print(2, "Valist_readPDB: Can't find z!\n");
00408         return VRC_FAILURE;
00409     }
00410
00411 #if 0 /* Set to 1 if you want to debug */
00412     Vnm_print(1, "Valist_readPDB: serial = %d\n", *serial);
00413     Vnm_print(1, "Valist_readPDB: atomName = %s\n", atomName);
00414     Vnm_print(1, "Valist_readPDB: resName = %s\n", resName);
00415     Vnm_print(1, "Valist_readPDB: resSeq = %d\n", *resSeq);
00416     Vnm_print(1, "Valist_readPDB: pos = (%g, %g, %g)\n",
00417               *x, *y, *z);
00418 #endif
00419
00420     return VRC_SUCCESS;
00421 }
00422
00423 /* Get a the next available atom storage location, increasing the storage
00424 * space if necessary. Return VNULL if something goes wrong. */
00425 VPRIVATE Vatom* Valist_getAtomStorage(
00426     Valist *thee,
00427     Vatom **plist, /* Pointer to existing list of atoms */
00428     int *pnlist, /* Size of existing list, may be changed */
00429     int *pnatoms /* Existing number of atoms in list; incremented
00430                   before exit */
00431 )
00432
00433     Vatom *oldList, *newList, *theList;
00434     Vatom *oldAtom, *newAtom;
00435     int iatom, inext, oldLength, newLength, natoms;
00436
00437     newList = VNULL;
00438
00439     /* See if we need more space */
00440     if (*pnatoms >= *pnlist) {
00441
00442         /* Double the storage space */
00443         oldLength = *pnlist;
00444         newLength = 2*oldLength;
00445         newList = Vmem_malloc(thee->vmem, newLength, sizeof(Vatom));
00446         oldList = *plist;
00447
00448         /* Check the allocation */
00449         if (newList == VNULL) {
00450             Vnm_print(2, "Valist_readPDB: failed to allocate space for %d (Vatom
00451 )s!\n", newLength);
00452             return VNULL;
00453         }
00454
00455         /* Copy the atoms over */
00456         natoms = *pnatoms;

```

```

00456         for (iatom=0; iatom<natoms; iatom++) {
00457             oldAtom = &(oldList[iatom]);
00458             newAtom = &(newList[iatom]);
00459             Vatom_copyTo(oldAtom, newAtom);
00460             Vatom_dtor2(oldAtom);
00461         }
00462
00463         /* Free the old list */
00464         Vmem_free(thee->vmem, oldLength, sizeof(Vatom), (void **)plist);
00465
00466         /* Copy new list to plist */
00467         *plist = newList;
00468         *pnlist = newLength;
00469     }
00470
00471     theList = *plist;
00472     inext = *pnatoms;
00473
00474     /* Get the next available spot and increment counters */
00475     newAtom = &(theList[inext]);
00476     *pnatoms = inext + 1;
00477
00478     return newAtom;
00479 }
00480
00481 VPRIVATE Vrc_Codes Valist_setAtomArray(Valist *thee,
00482                                         Vatom **plist, /* Pointer to list of atoms to store */
00483                                         int nlist, /* Length of list */
00484                                         int natoms /* Number of real atom entries in list */
00485                                         ) {
00486
00487     Vatom *list, *newAtom, *oldAtom;
00488     int i;
00489
00490     list = *plist;
00491
00492     /* Allocate necessary space */
00493     thee->number = 0;
00494     thee->atoms = Vmem_malloc(thee->vmem, natoms, sizeof(Vatom));
00495     if (thee->atoms == VNULL) {
00496         Vnm_print(2, "Valist_readPDB: Unable to allocate space for %d (Vatom)s!\\" n",
00497                   natoms);
00498         return VRC_FAILURE;
00499     }
00500     thee->number = natoms;
00501
00502     /* Copy over data */
00503     for (i=0; i<thee->number; i++) {
00504         newAtom = &(thee->atoms[i]);
00505         oldAtom = &(list[i]);
00506         Vatom_copyTo(oldAtom, newAtom);
00507         Vatom_dtor2(oldAtom);
00508     }
00509
00510     /* Free old array */
00511     Vmem_free(thee->vmem, nlist, sizeof(Vatom), (void **)plist);

```

```

00512
00513     return VRC_SUCCESS;
00514 }
00515
00516 VPUBLIC Vrc_Codes Valist_readPDB(Valist *thee, Vparam *param, Vio *sock) {
00517
00518     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00519      * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00520
00521     Vatom *atoms = VNULL;
00522     Vatom *nextAtom = VNULL;
00523     Vparam_AtomData *atomData = VNULL;
00524
00525     char tok[VMAX_BUFSIZE];
00526     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00527
00528     int nlist, natoms, serial, resSeq;
00529
00530     double x, y, z, charge, radius, epsilon;
00531     double pos[3];
00532
00533     if (thee == VNULL) {
00534         Vnm_print(2, "Valist_readPDB: Got NULL pointer when reading PDB file!\n");
00535         VASSERT(0);
00536     }
00537     thee->number = 0;
00538
00539     Vio_setWhiteChars(sock, Valist_whiteChars);
00540     Vio_setCommChars(sock, Valist_commChars);
00541
00542     /* Allocate some initial space for the atoms */
00543     nlist = 200;
00544     atoms = Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00545
00546     natoms = 0;
00547     /* Read until we run out of lines */
00548     while (Vio_scanf(sock, "%s", tok) == 1) {
00549
00550         /* Parse only ATOM/HETATOM fields */
00551         if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
00552             (Vstring_strcasecmp(tok, "HETATM") == 0)) {
00553
00554             /* Read ATOM/HETATOM field of PDB through the X/Y/Z fields */
00555             if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00556                                         resName, &resSeq, &x, &y, &z) == VRC_FAILURE) {
00557                 Vnm_print(2, "Valist_readPDB: Error parsing atom %d!\n",
00558                           serial);
00559                 return VRC_FAILURE;
00560             }
00561
00562             /* Try to find the parameters. */
00563             atomData = Vparam_getAtomData(param, resName, atomName);
00564             if (atomData == VNULL) {
00565                 Vnm_print(2, "Valist_readPDB: Couldn't find parameters for \
00566 atom = %s, residue = %s\n", atomName, resName);
00567                 return VRC_FAILURE;
00568             }

```

```

00569         charge = atomData->charge;
00570         radius = atomData->radius;
00571         epsilon = atomData->epsilon;
00572
00573         /* Get pointer to next available atom position */
00574         nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00575         if (nextAtom == VNULL) {
00576             Vnm_print(2, "Valist_readPDB: Error in allocating spacing for at
00577             oms!\n");
00578             return VRC_FAILURE;
00579         }
00580
00581         /* Store the information */
00582         pos[0] = x; pos[1] = y; pos[2] = z;
00583         Vatom_setPosition(nextAtom, pos);
00584         Vatom_setCharge(nextAtom, charge);
00585         Vatom_setRadius(nextAtom, radius);
00586         Vatom_setEpsilon(nextAtom, epsilon);
00587         Vatom_setAtomID(nextAtom, natoms-1);
00588         Vatom_setResName(nextAtom, resName);
00589         Vatom_setAtomName(nextAtom, atomName);
00590
00591     } /* if ATOM or HETATOM */
00592     /* while we haven't run out of tokens */
00593
00594     Vnm_print(0, "Valist_readPDB: Counted %d atoms\n", natoms);
00595     fflush(stdout);
00596
00597     /* Store atoms internally */
00598     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00599         Vnm_print(2, "Valist_readPDB: unable to store atoms!\n");
00600         return VRC_FAILURE;
00601     }
00602
00603     return Valist_getStatistics(thee);
00604
00605 }
00606
00607 VPUBLIC Vrc_Codes Valist_readPQR(Valist *thee, Vparam *params, Vio *sock) {
00608
00609     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00610      * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00611
00612
00613     Vatom *atoms = VNULL;
00614     Vatom *nextAtom = VNULL;
00615     Vparam_AtomData *atomData = VNULL;
00616
00617     char tok[VMAX_BUFSIZE];
00618     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00619
00620     int use_params = 0;
00621     int nlist, natoms, serial, resSeq;
00622
00623     double x, y, z, charge, radius, epsilon;
00624     double pos[3];

```

```

00625
00626     epsilon = 0.0;
00627
00628     if (thee == VNULL) {
00629         Vnm_print(2, "Valist_readPQR: Got NULL pointer when reading PQR file!\n");
00630         VASSERT(0);
00631     }
00632     thee->number = 0;
00633
00634     Vio_setWhiteChars(sock, Valist_whiteChars);
00635     Vio_setCommChars(sock, Valist_commChars);
00636
00637     /* Allocate some initial space for the atoms */
00638     nlist = 200;
00639     atoms = Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00640
00641 /* Check if we are using a parameter file or not */
00642 if (params != VNULL) use_params = 1;
00643
00644     natoms = 0;
00645     /* Read until we run out of lines */
00646     while (Vio_scanf(sock, "%s", tok) == 1) {
00647
00648         /* Parse only ATOM/HETATOM fields */
00649         if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
00650             (Vstring_strcasecmp(tok, "HETATOM") == 0)) {
00651
00652             /* Read ATOM/HETATOM field of PDB through the X/Y/Z fields */
00653             if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00654                 resName, &resSeq, &x, &y, &z) == VRC_FAILURE) {
00655                 Vnm_print(2, "Valist_readPQR: Error parsing atom %d!\n", serial);
00656
00657                 Vnm_print(2, "Please double check this atom in the pqr file, e.g.
00658 , make sure there are no concatenated fields.\n");
00659             }
00660
00661             /* Read Q/R fields */
00662             if (Valist_readPDBChargeRadius(thee, sock, &charge, &radius) ==
00663                 VRC_FAILURE) {
00664                 Vnm_print(2, "Valist_readPQR: Error parsing atom %d!\n",
00665                     serial);
00666                 Vnm_print(2, "Please double check this atom in the pqr file, e.g.
00667 , make sure there are no concatenated fields.\n");
00668             }
00669             if (use_params) {
00670                 /* Try to find the parameters. */
00671                 atomData = Vparam_getAtomData(params, resName, atomName);
00672                 if (atomData == VNULL) {
00673                     Vnm_print(2, "Valist_readPDB: Couldn't find parameters for \
00674 atom = %s, residue = %s\n", atomName, resName);
00675                     return VRC_FAILURE;
00676                 }
00677                 charge = atomData->charge;
00678                 radius = atomData->radius;

```

```

00678     epsilon = atomData->epsilon;
00679 }
00680
00681         /* Get pointer to next available atom position */
00682         nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00683         if (nextAtom == VNULL) {
00684             Vnm_print(2, "Valist_readPQR: Error in allocating spacing for at
00685             oms!\n");
00686             return VRC_FAILURE;
00687         }
00688         /* Store the information */
00689         pos[0] = x; pos[1] = y; pos[2] = z;
00690         Vatom_setPosition(nextAtom, pos);
00691         Vatom_setCharge(nextAtom, charge);
00692         Vatom_setRadius(nextAtom, radius);
00693         Vatom_setEpsilon(nextAtom, epsilon);
00694         Vatom_setAtomID(nextAtom, natoms-1);
00695         Vatom_setResName(nextAtom, resName);
00696         Vatom_setAtomName(nextAtom, atomName);
00697
00698     } /* if ATOM or HETATOM */
00699 } /* while we haven't run out of tokens */
00700
00701 Vnm_print(0, "Valist_readPQR: Counted %d atoms\n", natoms);
00702 fflush(stdout);
00703
00704 /* Store atoms internally */
00705 if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00706     Vnm_print(2, "Valist_readPDB: unable to store atoms!\n");
00707     return VRC_FAILURE;
00708 }
00709
00710 return Valist_getStatistics(thee);
00711
00712
00713 }
00714
00715 VPUBLIC Vrc_Codes Valist_readXML(Valist *thee, Vparam *params, Vio *sock) {
00716
00717     Vatom *atoms = VNULL;
00718     Vatom *nextAtom = VNULL;
00719
00720     char tok[VMAX_BUFSIZE];
00721     char endtag[VMAX_BUFSIZE];
00722
00723     int nlist, natoms;
00724     int xset, yset, zset, chgset, radset;
00725
00726     double x, y, z, charge, radius, dtmp;
00727     double pos[3];
00728
00729     if (thee == VNULL) {
00730         Vnm_print(2, "Valist_readXML: Got NULL pointer when reading XML file!\n");
00731         VASSERT(0);
00732     }
00733     thee->number = 0;

```

```

00734
00735     Vio_setWhiteChars(sock, Valist_xmlwhiteChars);
00736     Vio_setCommChars(sock, Valist_commChars);
00737
00738     /* Allocate some initial space for the atoms */
00739     nlist = 200;
00740     atoms = Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00741
00742     /* Initialize some variables */
00743     natoms = 0;
00744     xset = 0;
00745     yset = 0;
00746     zset = 0;
00747     chgset = 0;
00748     radset = 0;
00749     strcpy(endtag, "/");
00750
00751     if (params == VNULL) {
00752         Vnm_print(1, "\nValist_readXML: Warning Warning Warning Warning\n");
00753         Vnm_print(1, "Valist_readXML: The use of XML input files with parameter\n");
00754         Vnm_print(1, "Valist_readXML: files is currently not supported.\n");
00755         Vnm_print(1, "Valist_readXML: Warning Warning Warning Warning\n\n");
00756     }
00757
00758     /* Read until we run out of lines */
00759     while (Vio_scanf(sock, "%s", tok) == 1) {
00760
00761         /* The first tag taken is the start tag - save it to detect end */
00762         if (Vstring_strcasecmp(endtag, "/") == 0) strcat(endtag, tok);
00763
00764         if (Vstring_strcasecmp(tok, "x") == 0) {
00765             Vio_scanf(sock, "%s", tok);
00766             if (sscanf(tok, "%lf", &dtmp) != 1) {
00767                 Vnm_print(2, "Valist_readXML: Unexpected token (%s) while \
00768 reading x!\n", tok);
00769                 return VRC_FAILURE;
00770             }
00771             x = dtmp;
00772             xset = 1;
00773         } else if (Vstring_strcasecmp(tok, "y") == 0) {
00774             Vio_scanf(sock, "%s", tok);
00775             if (sscanf(tok, "%lf", &dtmp) != 1) {
00776                 Vnm_print(2, "Valist_readXML: Unexpected token (%s) while \
00777 reading y!\n", tok);
00778                 return VRC_FAILURE;
00779             }
00780             y = dtmp;
00781             yset = 1;
00782         } else if (Vstring_strcasecmp(tok, "z") == 0) {
00783             Vio_scanf(sock, "%s", tok);
00784             if (sscanf(tok, "%lf", &dtmp) != 1) {
00785                 Vnm_print(2, "Valist_readXML: Unexpected token (%s) while \
00786 reading z!\n", tok);
00787                 return VRC_FAILURE;
00788             }
00789             z = dtmp;
00790             zset = 1;

```

```

00791         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00792             Vio_scanf(sock, "%s", tok);
00793             if (sscanf(tok, "%lf", &dtmp) != 1) {
00794                 Vnm_print(2, "Valist_readXML: Unexpected token (%s) while \
00795 reading charge!\n", tok);
00796                 return VRC_FAILURE;
00797             }
00798             charge = dtmp;
00799             chgset = 1;
00800         } else if (Vstring_strcasecmp(tok, "radius") == 0) {
00801             Vio_scanf(sock, "%s", tok);
00802             if (sscanf(tok, "%lf", &dtmp) != 1) {
00803                 Vnm_print(2, "Valist_readXML: Unexpected token (%s) while \
00804 reading radius!\n", tok);
00805                 return VRC_FAILURE;
00806             }
00807             radius = dtmp;
00808             radset = 1;
00809         } else if (Vstring_strcasecmp(tok, "/atom") == 0) {
00810             /* Get pointer to next available atom position */
00811             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00812             if (nextAtom == VNULL) {
00813                 Vnm_print(2, "Valist_readXML: Error in allocating spacing for at \
00814 oms!\n");
00815                 return VRC_FAILURE;
00816             }
00817             if (xset && yset && zset && chgset && radset) {
00818                 /* Store the information */
00819                 pos[0] = x; pos[1] = y; pos[2] = z;
00820                 Vatom_setPosition(nextAtom, pos);
00821                 Vatom_setCharge(nextAtom, charge);
00822                 Vatom_setRadius(nextAtom, radius);
00823                 Vatom_setAtomID(nextAtom, natoms-1);
00824
00825                 /* Reset the necessary flags */
00826                 xset = 0;
00827                 yset = 0;
00828                 zset = 0;
00829                 chgset = 0;
00830                 radset = 0;
00831             } else {
00832                 Vnm_print(2, "Valist_readXML: Missing field(s) in atom tag:\n");
00833             }
00834         ;
00835         if (!xset) Vnm_print(2, "\tx value not set!\n");
00836         if (!yset) Vnm_print(2, "\ty value not set!\n");
00837         if (!zset) Vnm_print(2, "\tz value not set!\n");
00838         if (!chgset) Vnm_print(2, "\tcharge value not set!\n");
00839         if (!radset) Vnm_print(2, "\tradius value not set!\n");
00840         return VRC_FAILURE;
00841     }
00842     } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
00843 }
00844
00845 Vnm_print(0, "Valist_readXML: Counted %d atoms\n", natoms);

```

```

00846     fflush(stdout);
00847
00848     /* Store atoms internally */
00849     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00850         Vnm_print(2, "Valist_readXML: unable to store atoms!\n");
00851         return VRC_FAILURE;
00852     }
00853
00854     return Valist_getStatistics(thee);
00855
00856 }
00857
00858 /* Load up Valist with various statistics */
00859 VPUBLIC Vrc_Codes Valist_getStatistics(Valist *thee) {
00860
00861     Vatom *atom;
00862     int i, j;
00863
00864     if (thee == VNULL) {
00865         Vnm_print(2, "Valist_getStatistics: Got NULL pointer when loading up Valist wi
th various statistics!\n");
00866         VASSERT(0);
00867     }
00868
00869     thee->center[0] = 0.;
00870     thee->center[1] = 0.;
00871     thee->center[2] = 0.;
00872     thee->maxrad = 0.;
00873     thee->charge = 0.;
00874
00875     if (thee->number == 0) return VRC_FAILURE;
00876
00877     /* Reset stat variables */
00878     atom = &(thee->atoms[0]);
00879     for (i=0; i<3; i++) {
00880         thee->maxcrd[i] = thee->mincrd[i] = atom->position[i];
00881     }
00882     thee->maxrad = atom->radius;
00883     thee->charge = 0.0;
00884
00885     for (i=0; i<thee->number; i++) {
00886
00887         atom = &(thee->atoms[i]);
00888         for (j=0; j<3; j++) {
00889             if (atom->position[j] < thee->mincrd[j])
00890                 thee->mincrd[j] = atom->position[j];
00891             if (atom->position[j] > thee->maxcrd[j])
00892                 thee->maxcrd[j] = atom->position[j];
00893         }
00894         if (atom->radius > thee->maxrad) thee->maxrad = atom->radius;
00895         thee->charge = thee->charge + atom->charge;
00896     }
00897
00898     thee->center[0] = 0.5*(thee->maxcrd[0] + thee->mincrd[0]);
00899     thee->center[1] = 0.5*(thee->maxcrd[1] + thee->mincrd[1]);
00900     thee->center[2] = 0.5*(thee->maxcrd[2] + thee->mincrd[2]);
00901

```

```

00902 Vnm_print(0, "Valist_getStatistics: Max atom coordinate: (%g, %g, %g)\n",
00903     thee->maxcrd[0], thee->maxcrd[1], thee->maxcrd[2]);
00904 Vnm_print(0, "Valist_getStatistics: Min atom coordinate: (%g, %g, %g)\n",
00905     thee->mincrd[0], thee->mincrd[1], thee->mincrd[2]);
00906 Vnm_print(0, "Valist_getStatistics: Molecule center: (%g, %g, %g)\n",
00907     thee->center[0], thee->center[1], thee->center[2]);
00908
00909     return VRC_SUCCESS;
00910 }

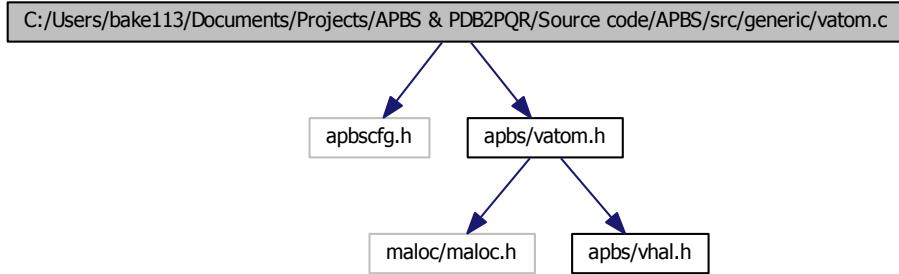
```

10.65 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vatom.c File Reference

Class Vatom methods.

```
#include "apbscfg.h"
#include "apbs/vatom.h"
```

Include dependency graph for vatom.c:



Functions

- VPUBLIC double * **Vatom_getPosition** (**Vatom** *thee)
Get atomic position.
- VPUBLIC double **Vatom_getPartID** (**Vatom** *thee)
Get partition ID.
- VPUBLIC void **Vatom_setPartID** (**Vatom** *thee, int partID)
Set partition ID.

- VPUBLIC double `Vatom_getAtomID` (`Vatom *thee`)

Get atom ID.
- VPUBLIC void `Vatom_setAtomID` (`Vatom *thee, int atomID`)

Set atom ID.
- VPUBLIC void `Vatom_setRadius` (`Vatom *thee, double radius`)

Set atomic radius.
- VPUBLIC double `Vatom_getRadius` (`Vatom *thee`)

Get atomic position.
- VPUBLIC void `Vatom_setCharge` (`Vatom *thee, double charge`)

Set atomic charge.
- VPUBLIC double `Vatom_getCharge` (`Vatom *thee`)

Get atomic charge.
- VPUBLIC unsigned long int `Vatom_memChk` (`Vatom *thee`)

Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC `Vatom * Vatom_ctor` ()

Constructor for the Vatom class.
- VPUBLIC int `Vatom_ctor2` (`Vatom *thee`)

FORTRAN stub constructor for the Vatom class.
- VPUBLIC void `Vatom_dtor` (`Vatom **thee`)

Object destructor.
- VPUBLIC void `Vatom_dtor2` (`Vatom *thee`)

FORTRAN stub object destructor.
- VPUBLIC void `Vatom_setPosition` (`Vatom *thee, double position[3]`)

Set the atomic position.
- VPUBLIC void `Vatom_copyTo` (`Vatom *thee, Vatom *dest`)

Copy information to another atom.
- VPUBLIC void `Vatom_copyFrom` (`Vatom *thee, Vatom *src`)

Copy information to another atom.
- VPUBLIC void `Vatom_setResName` (`Vatom *thee, char resName[VMAX_RECLEN]`)

Set residue name.
- VPUBLIC void `Vatom_getResName` (`Vatom *thee, char resName[VMAX_RECLEN]`)

Retrieve residue name.
- VPUBLIC void `Vatom_setAtomName` (`Vatom *thee, char atomName[VMAX_RECLEN]`)

Set atom name.
- VPUBLIC void `Vatom_getAtomName` (`Vatom *thee, char atomName[VMAX_RECLEN]`)

Retrieve atom name.

10.65.1 Detailed Description

Class Vatom methods.

Author

Nathan Baker

Version

Id:

[vatom.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vatom.c](#).

10.66 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vatom.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vatom.h"
00059
00060 VEMBED(rcsid="$Id: vatom.c 1667 2011-12-02 23:22:02Z pcellis $")
00061
00062 #if !defined(VINLINE_VATOM)
00063
00064 VPUBLIC double *Vatom_getPosition(Vatom *thee) {
00065
00066     VASSERT(thee != VNULL);
00067     return thee->position;
00068
00069 }
00070
00071 VPUBLIC double Vatom_getPartID(Vatom *thee) {
00072
00073     VASSERT(thee != VNULL);
00074     return thee->partID;
00075
00076 }
00077
00078 VPUBLIC void Vatom_setPartID(Vatom *thee, int partID) {
00079
00080     VASSERT(thee != VNULL);
00081     thee->partID = (double)partID;
00082
00083 }
00084
00085 VPUBLIC double Vatom_getAtomID(Vatom *thee) {
00086
00087     VASSERT(thee != VNULL);
00088     return thee->atomID;
00089
00090 }
00091
00092 VPUBLIC void Vatom_setAtomID(Vatom *thee, int atomID) {
00093
00094     VASSERT(thee != VNULL);

```

```

00095     thee->atomID = atomID;
00096
00097 }
00098
00099 VPUBLIC void Vatom_setRadius(Vatom *thee, double radius) {
00100
00101     VASSERT(thee != VNULL);
00102     thee->radius = radius;
00103
00104 }
00105
00106 VPUBLIC double Vatom_getRadius(Vatom *thee) {
00107
00108     VASSERT(thee != VNULL);
00109     return thee->radius;
00110
00111 }
00112
00113 VPUBLIC void Vatom_setCharge(Vatom *thee, double charge) {
00114
00115     VASSERT(thee != VNULL);
00116     thee->charge = charge;
00117
00118 }
00119
00120 VPUBLIC double Vatom_getCharge(Vatom *thee) {
00121
00122     VASSERT(thee != VNULL);
00123     return thee->charge;
00124
00125 }
00126
00127 VPUBLIC unsigned long int Vatom_memChk(Vatom *thee) { return sizeof(Vatom); }
00128
00129 #endif /* if !defined(VINLINE_VATOM) */
00130
00131 VPUBLIC Vatom* Vatom_ctor() {
00132
00133     /* Set up the structure */
00134     Vatom *thee = VNULL;
00135     thee = (Vatom *)Vmem_malloc( VNULL, 1, sizeof(Vatom) );
00136     VASSERT( thee != VNULL );
00137     VASSERT( Vatom_ctor2(thee) );
00138
00139     return thee;
00140 }
00141
00142 VPUBLIC int Vatom_ctor2(Vatom *thee) {
00143     thee->partID = -1;
00144     return 1;
00145 }
00146
00147 VPUBLIC void Vatom_dtor(Vatom **thee) {
00148     if ((*thee) != VNULL) {
00149         Vatom_dtor2(*thee);
00150         Vmem_free(VNULL, 1, sizeof(Vatom), (void **)thee);
00151         (*thee) = VNULL;

```

```
00152     }
00153 }
00154
00155 VPUBLIC void Vatom_dtor2(Vatom *thee) { ; }
00156
00157 VPUBLIC void Vatom_setPosition(Vatom *thee, double position[3]) {
00158
00159     VASSERT(thee != VNULL);
00160     (thee->position)[0] = position[0];
00161     (thee->position)[1] = position[1];
00162     (thee->position)[2] = position[2];
00163
00164 }
00165
00166 VPUBLIC void Vatom_copyTo(Vatom *thee, Vatom *dest) {
00167
00168     VASSERT(thee != VNULL);
00169     VASSERT(dest != VNULL);
00170
00171     memcpy(dest, thee, sizeof(Vatom));
00172
00173 }
00174
00175 VPUBLIC void Vatom_copyFrom(Vatom *thee, Vatom *src) {
00176
00177     Vatom_copyTo(src, thee);
00178
00179 }
00180
00181 VPUBLIC void Vatom_setResName(Vatom *thee, char resName[VMAX_RECLEN]) {
00182
00183     VASSERT(thee != VNULL);
00184     strcpy(thee->resName, resName);
00185
00186 }
00187
00188 VPUBLIC void Vatom_getResName(Vatom *thee, char resName[VMAX_RECLEN]) {
00189
00190
00191     VASSERT(thee != VNULL);
00192     strcpy(resName, thee->resName);
00193
00194 }
00195
00196 VPUBLIC void Vatom_setAtomName(Vatom *thee, char atomName[VMAX_RECLEN]) {
00197
00198     VASSERT(thee != VNULL);
00199     strcpy(thee->atomName, atomName);
00200
00201 }
00202
00203 VPUBLIC void Vatom_getAtomName(Vatom *thee, char atomName[VMAX_RECLEN]) {
00204
00205     VASSERT(thee != VNULL);
00206     strcpy(atomName, thee->atomName);
00207
00208 }
```

```
00209
00210 #if defined(WITH_TINKER)
00211
00212 VPUBLIC void Vatom_setDipole(Vatom *thee, double dipole[3]) {
00213
00214     VASSERT(thee != VNULL);
00215     (thee->dipole)[0] = dipole[0];
00216     (thee->dipole)[1] = dipole[1];
00217     (thee->dipole)[2] = dipole[2];
00218 }
00219
00220
00221 VPUBLIC void Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]) {
00222
00223     int i;
00224     VASSERT(thee != VNULL);
00225     for (i=0; i<9; i++) (thee->quadrupole)[i] = quadrupole[i];
00226 }
00227
00228 VPUBLIC void Vatom_setInducedDipole(Vatom *thee, double dipole[3]) {
00229
00230     VASSERT(thee != VNULL);
00231     (thee->inducedDipole)[0] = dipole[0];
00232     (thee->inducedDipole)[1] = dipole[1];
00233     (thee->inducedDipole)[2] = dipole[2];
00234 }
00235
00236 VPUBLIC void Vatom_setNLIInducedDipole(Vatom *thee, double dipole[3]) {
00237
00238     VASSERT(thee != VNULL);
00239     (thee->nLIInducedDipole)[0] = dipole[0];
00240     (thee->nLIInducedDipole)[1] = dipole[1];
00241     (thee->nLIInducedDipole)[2] = dipole[2];
00242
00243 }
00244
00245 VPUBLIC double *Vatom_getDipole(Vatom *thee) {
00246
00247     VASSERT(thee != VNULL);
00248     return thee->dipole;
00249
00250 }
00251
00252 VPUBLIC double *Vatom_getQuadrupole(Vatom *thee) {
00253
00254     VASSERT(thee != VNULL);
00255     return thee->quadrupole;
00256
00257 }
00258
00259 VPUBLIC double *Vatom_getInducedDipole(Vatom *thee) {
00260
00261     VASSERT(thee != VNULL);
00262     return thee->inducedDipole;
00263
00264 }
00265
```

```

00266 VPUBLIC double *Vatom_getNLInducedDipole(Vatom *thee) {
00267
00268     VASSERT(thee != VNULL);
00269     return thee->nlInducedDipole;
00270
00271 }
00272
00273 #endif /* if defined(WITH_TINKER) */

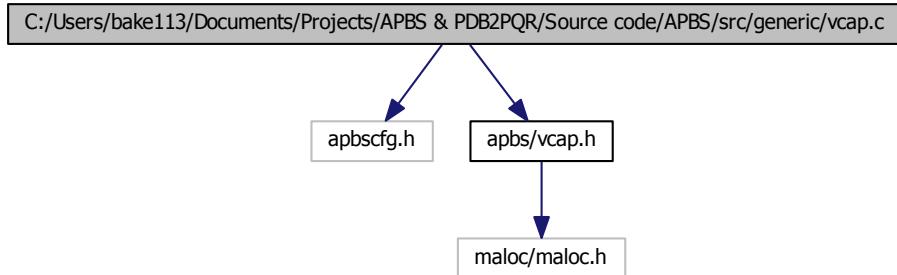
```

10.67 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vcap.c File Reference

Class Vcap methods.

```
#include "apbscfg.h"
#include "apbs/vcap.h"
```

Include dependency graph for vcap.c:



Functions

- VPUBLIC double **Vcap_exp** (double x, int *ichop)
Provide a capped exp() function.
- VPUBLIC double **Vcap_sinh** (double x, int *ichop)
Provide a capped sinh() function.
- VPUBLIC double **Vcap_cosh** (double x, int *ichop)
Provide a capped cosh() function.

10.67.1 Detailed Description

Class Vcap methods.

Author

Nathan Baker

Version

Id:

[vcap.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vcap.c](#).

10.68 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vcap.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vcap.h"
00059
00060 VPUBLIC double Vcap_exp(double x, int *ichop) {
00061
00062     /* The two chopped arguments */
00063     if (x > EXPMAX) {
00064         (*ichop) = 1;
00065         return VEXP(EXPMAX);
00066     } else if (x < EXPMIN) {
00067         (*ichop) = 1;
00068         return VEXP(EXPMIN);
00069     }
00070
00071     /* The normal EXP */
00072     (*ichop) = 0;
00073     return VEXP(x);
00074 }
00075
00076 VPUBLIC double Vcap_sinh(double x, int *ichop) {
00077
00078     /* The two chopped arguments */
00079     if (x > EXPMAX) {
00080         (*ichop) = 1;
00081         return VSINH(EXPMAX);
00082     } else if (x < EXPMIN) {
00083         (*ichop) = 1;
00084         return VSINH(EXPMIN);
00085     }
00086
00087     /* The normal SINH */
00088     (*ichop) = 0;
00089     return VSINH(x);
00090 }
00091
00092 VPUBLIC double Vcap_cosh(double x, int *ichop) {
00093
00094     /* The two chopped arguments */

```

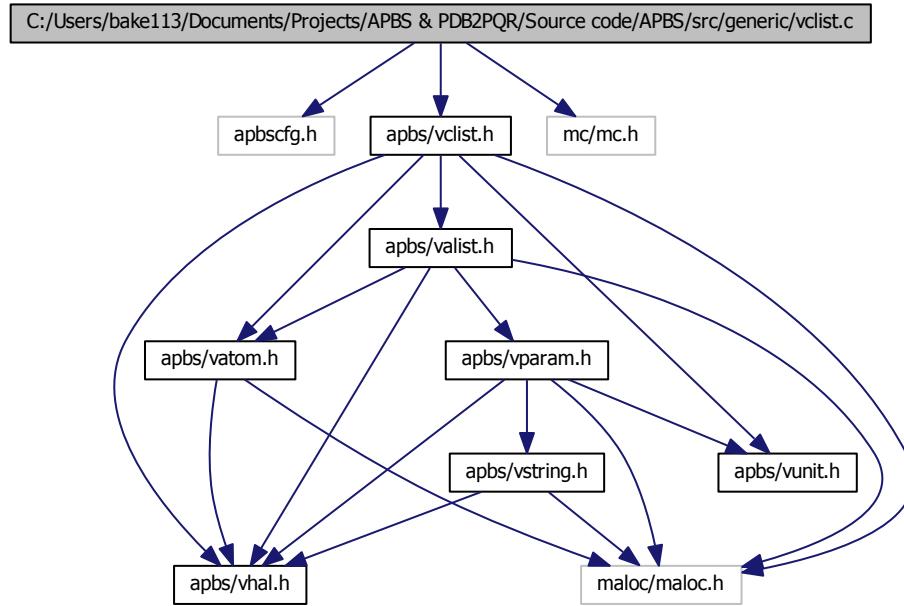
```
00095     if (x > EXPMAX) {
00096         (*ichop) = 1;
00097         return VCOSH(EXPMAX);
00098     } else if (x < EXPMIN) {
00099         (*ichop) = 1;
00100         return VCOSH(EXPMIN);
00101     }
00102     /* The normal COSH */
00103     (*ichop) = 0;
00104     return VCOSH(x);
00105 }
00106 }
```

10.69 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vclist.c File Reference

Class Vclist methods.

```
#include "apbscfg.h"
#include "apbs/vclist.h"
#include "mc/mc.h"
```

Include dependency graph for vclist.c:



Defines

- #define **VCLIST_INFLATE** 1.42

Functions

- VPUBLIC unsigned long int **Vclist_memChk** (**Vclist** *thee)
Get number of bytes in this object and its members.
- VPUBLIC double **Vclist_maxRadius** (**Vclist** *thee)
Get the max probe radius value (in A) the cell list was constructed with.
- VPUBLIC **Vclist** * **Vclist_ctor** (**Valist** *alist, double max_radius, int npts[VAPBS_DIM], **Vclist_DomainMode** mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])
Construct the cell list object.

- VPRIVATE void **Vclist_getMolDims** (**Vclist** *thee, double lower_corner[VAPBS_-DIM], double upper_corner[VAPBS_DIM], double *r_max)
- VPRIVATE Vrc_Codes **Vclist_setupGrid** (**Vclist** *thee)
- VPRIVATE Vrc_Codes **Vclist_storeParms** (**Vclist** *thee, **Valist** *alist, double max_radius, int npts[VAPBS_DIM], **Vclist_DomainMode** mode, double lower_corner[VAPBS_-DIM], double upper_corner[VAPBS_DIM])
- VPRIVATE void **Vclist_gridSpan** (**Vclist** *thee, **Vatom** *atom, int imin[VAPBS_-DIM], int imax[VAPBS_DIM])
- VPRIVATE int **Vclist_arrayIndex** (**Vclist** *thee, int i, int j, int k)
- VPRIVATE Vrc_Codes **Vclist_assignAtoms** (**Vclist** *thee)
- VPUBLIC Vrc_Codes **Vclist_ctor2** (**Vclist** *thee, **Valist** *alist, double max_radius, int npts[VAPBS_DIM], **Vclist_DomainMode** mode, double lower_corner[VAPBS_-DIM], double upper_corner[VAPBS_DIM])
FORTRAN stub to construct the cell list object.
- VPUBLIC void **Vclist_dtor** (**Vclist** **thee)
Destroy object.
- VPUBLIC void **Vclist_dtor2** (**Vclist** *thee)
FORTRAN stub to destroy object.
- VPUBLIC **VclistCell** * **Vclist_getCell** (**Vclist** *thee, double pos[VAPBS_DIM])
Return cell corresponding to specified position or return VNULL.
- VPUBLIC **VclistCell** * **VclistCell_ctor** (int natoms)
Allocate and construct a cell list cell object.
- VPUBLIC Vrc_Codes **VclistCell_ctor2** (**VclistCell** *thee, int natoms)
Construct a cell list object.
- VPUBLIC void **VclistCell_dtor** (**VclistCell** **thee)
Destroy object.
- VPUBLIC void **VclistCell_dtor2** (**VclistCell** *thee)
FORTRAN stub to destroy object.

10.69.1 Detailed Description

Class Vclist methods.

Author

Nathan Baker

Version

Id:

[vclist.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*
```

Definition in file [vclist.c](#).

10.70 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vclist.c

```
00001
00057 #include "apbscfg.h"
00058 #include "apbs/vclist.h"
00059
00060 #if defined(HAVE_MC_H)
00061 #include "mc/mc.h"
00062 #endif
00063
00064 VEMBED(rcsid="$Id: vclist.c 1667 2011-12-02 23:22:02Z pcellis $")
00065
00066 #if !defined(VINLINE_VCLIST)
00067
00068 VPUBLIC unsigned long int Vclist_memChk(Vclist *thee) {
00069     if (thee == VNULL) return 0;
00070     return Vmem_bytes(thee->vmem);
00071 }
00072
00073 VPUBLIC double Vclist_maxRadius(Vclist *thee) {
00074     VASSERT(thee != VNULL);
00075     return thee->max_radius;
00076 }
00077
00078 #endif /* if !defined(VINLINE_VCLIST) */
00079
00080 VPUBLIC Vclist* Vclist_ctor(Valist *alist, double max_radius,
00081     int npts[VAPBS_DIM], Vclist_DomainMode mode,
00082     double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]) {
00083
00084     Vclist *thee = VNULL;
00085
00086     /* Set up the structure */
00087     thee = Vmem_malloc(VNULL, 1, sizeof(Vclist));
00088     VASSERT( thee != VNULL );
00089     VASSERT( Vclist_ctor2(thee, alist, max_radius, npts, mode, lower_corner,
00090         upper_corner) == VRC_SUCCESS );
00091     return thee;
00092 }
00093
00094 /* Get the dimensions of the molecule stored in thee->alist */
00095 VPRIVATE void Vclist_getMolDims(
00096     Vclist *thee,
00097     double lower_corner[VAPBS_DIM], /* Set to lower corner of molecule */
00098     double upper_corner[VAPBS_DIM], /* Set to lower corner of molecule */
00099     double *r_max /* Set to max atom radius */
00100 )
00101
00102     int i, j;
00103     double pos;
00104     Valist *alist;
00105     Vatom *atom;
00106
00107     alist = thee->alist;
00108
```

```

00109 /* Initialize */
00110 for (i=0; i<VAPBS_DIM; i++) {
00111     lower_corner[i] = VLARGE;
00112     upper_corner[i] = -VLARGE;
00113 }
00114 *r_max = -1.0;
00115
00116 /* Check each atom */
00117 for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00118     atom = Valist_getAtom(alist, i);
00119     for (j=0; j<VAPBS_DIM; j++) {
00120         pos = (Vatom_getPosition(atom))[j];
00121         if (pos < lower_corner[j]) lower_corner[j] = pos;
00122         if (pos > upper_corner[j]) upper_corner[j] = pos;
00123     }
00124     if (Vatom_getRadius(atom) > *r_max) *r_max = Vatom_getRadius(atom);
00125 }
00126
00127 }
00128
00129 /* Setup lookup grid */
00130 VPRIVATE Vrc_Codes Vclist_setupGrid(Vclist *thee) {
00131
00132     /* Inflation factor ~ sqrt(2) */
00133     #define VCLIST_INFILATE 1.42
00134
00135     int i;
00136     double length[VAPBS_DIM], r_max;
00137
00138     /* Set up the grid corners */
00139     switch (thee->mode) {
00140         case CLIST_AUTO_DOMAIN:
00141             /* Get molecule dimensions */
00142             Vclist_getMolDims(thee, thee->lower_corner, thee->upper_corner,
00143                               &r_max);
00144             /* Set up grid spacings */
00145             for (i=0; i<VAPBS_DIM; i++) {
00146                 thee->upper_corner[i] = thee->upper_corner[i]
00147                     + VCLIST_INFILATE*(r_max+thee->max_radius);
00148                 thee->lower_corner[i] = thee->lower_corner[i]
00149                     - VCLIST_INFILATE*(r_max+thee->max_radius);
00150             }
00151             break;
00152         case CLIST_MANUAL_DOMAIN:
00153             /* Grid corners established in constructor */
00154             break;
00155         default:
00156             Vnm_print(2, "Vclist_setupGrid: invalid setup mode (%d)!\n",
00157                       thee->mode);
00158             return VRC_FAILURE;
00159     }
00160
00161     /* Set up the grid lengths and spacings */
00162     for (i=0; i<VAPBS_DIM; i++) {
00163         length[i] = thee->upper_corner[i] - thee->lower_corner[i];
00164         thee->spacs[i] = length[i]/((double)(thee->npts[i] - 1));
00165     }

```

```

00166     Vnm_print(0, "Vclist_setupGrid: Grid lengths = (%g, %g, %g)\n",
00167                 length[0], length[1], length[2]);
00168
00169     Vnm_print(0, "Vclist_setupGrid: Grid lower corner = (%g, %g, %g)\n",
00170                 (thee->lower_corner)[0], (thee->lower_corner)[1],
00171                 (thee->lower_corner)[2]);
00172
00173     return VRC_SUCCESS;
00174
00175     #undef VCLIST_INFLATE
00176 }
00177
00178 /* Check and store parameters passed to constructor */
00179 VPRIVATE Vrc_Codes Vclist_storeParms(Vclist *thee, Valist *alist,
00180             double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode,
00181             double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM] ) {
00182
00183     int i = 0;
00184
00185     if (alist == VNULL) {
00186         Vnm_print(2, "Vclist_ctor2: Got NULL Valist!\n");
00187         return VRC_FAILURE;
00188     } else thee->alist = alist;
00189
00190     thee->n = 1;
00191     for (i=0; i<VAPBS_DIM; i++) {
00192         if (npts[i] < 3) {
00193             Vnm_print(2,
00194                     "Vclist_ctor2: n[%d] (%d) must be greater than 2!\n",
00195                     i, npts[i]);
00196             return VRC_FAILURE;
00197         }
00198         thee->npts[i] = npts[i];
00199         thee->n *= npts[i];
00200     }
00201     Vnm_print(0, "Vclist_ctor2: Using %d x %d x %d hash table\n",
00202             npts[0], npts[1], npts[2]);
00203
00204     thee->mode = mode;
00205     switch (thee->mode) {
00206         case CLIST_AUTO_DOMAIN:
00207             Vnm_print(0, "Vclist_ctor2: automatic domain setup.\n");
00208             break;
00209         case CLIST_MANUAL_DOMAIN:
00210             Vnm_print(0, "Vclist_ctor2: manual domain setup.\n");
00211             Vnm_print(0, "Vclist_ctor2: lower corner = [ \n");
00212             for (i=0; i<VAPBS_DIM; i++) {
00213                 thee->lower_corner[i] = lower_corner[i];
00214                 Vnm_print(0, "%g ", lower_corner[i]);
00215             }
00216             Vnm_print(0, "]\n");
00217             Vnm_print(0, "Vclist_ctor2: upper corner = [ \n");
00218             for (i=0; i<VAPBS_DIM; i++) {
00219                 thee->upper_corner[i] = upper_corner[i];
00220                 Vnm_print(0, "%g ", upper_corner[i]);
00221             }
00222             Vnm_print(0, "]\n");

```

```

00223         break;
00224     default:
00225         Vnm_print(2, "Vclist_ctor2: invalid setup mode (%d)!\n", mode);
00226         return VRC_FAILURE;
00227     }
00228
00229     thee->max_radius = max_radius;
00230     Vnm_print(0, "Vclist_ctor2: Using %g max radius\n", max_radius);
00231
00232     return VRC_SUCCESS;
00233 }
00234
00235 /* Calculate the gridpoints an atom spans */
00236 VPRIIVATE void Vclist_gridSpan(Vclist *thee,
00237     Vatom *atom, /* Atom */
00238     int imin[VAPBS_DIM], /* Set to min grid indices */
00239     int imax[VAPBS_DIM] /* Set to max grid indices */
00240     ) {
00241
00242     int i;
00243     double *coord, dc, idc, rtot;
00244
00245     /* Get the position in the grid's frame of reference */
00246     coord = Vatom_getPosition(atom);
00247
00248     /* Get the range the atom radius + probe radius spans */
00249     rtot = Vatom_getRadius(atom) + thee->max_radius;
00250
00251     /* Calculate the range of grid points the inflated atom spans in the x
00252      * direction. */
00253     for (i=0; i<VAPBS_DIM; i++) {
00254         dc = coord[i] - (thee->lower_corner)[i];
00255         idc = (dc + rtot)/(thee->spacs[i]);
00256         imax[i] = (int)(ceil(idc));
00257         imax[i] = VMIN2(imax[i], thee->npts[i]-1);
00258         idc = (dc - rtot)/(thee->spacs[i]);
00259         imin[i] = (int)(floor(idc));
00260         imin[i] = VMAX2(imin[i], 0);
00261     }
00262
00263 }
00264
00265 /* Get the array index for a particular cell based on its i,j,k
00266   * coordinates */
00267 VPRIIVATE int Vclist_arrayIndex(Vclist *thee, int i, int j, int k) {
00268
00269     return (thee->npts[2])*(thee->npts[1])*i + (thee->npts[2])*j + k;
00270
00271 }
00272
00273
00274 /* Assign atoms to cells */
00275 VPRIIVATE Vrc_Codes Vclist_assignAtoms(Vclist *thee) {
00276
00277     int iatom, i, j, k, ui, inext;
00278     int imax[VAPBS_DIM], imin[VAPBS_DIM];
00279     int totatoms;

```

```

00280     Vatom *atom;
00281     VclistCell *cell;
00282
00283
00284     /* Find out how many atoms are associated with each grid point */
00285     totatoms = 0;
00286     for (iatom=0; iatom<Valist_getNumberAtoms (thee->alist); iatom++) {
00287
00288         /* Get grid span for atom */
00289         atom = Valist_getAtom(thee->alist, iatom);
00290         Vclist_gridSpan(thee, atom, imin, imax);
00291
00292         /* Now find and assign the grid points */
00293         VASSERT(VAPBS_DIM == 3);
00294         for ( i = imin[0]; i <= imax[0]; i++) {
00295             for ( j = imin[1]; j <= imax[1]; j++) {
00296                 for ( k = imin[2]; k <= imax[2]; k++) {
00297                     /* Get index to array */
00298                     ui = Vclist_arrayIndex(thee, i, j, k);
00299                     /* Increment number of atoms for this grid point */
00300                     cell = &(thee->cells[ui]);
00301                     (cell->natoms)++;
00302                     totatoms++;
00303                 }
00304             }
00305         }
00306     }
00307     Vnm_print(0, "Vclist_assignAtoms: Have %d atom entries\n", totatoms);
00308
00309     /* Allocate the space to store the pointers to the atoms */
00310     for (ui=0; ui<thee->n; ui++) {
00311         cell = &(thee->cells[ui]);
00312         if ( VclistCell_ctor2(cell, cell->natoms) == VRC_FAILURE ) {
00313             Vnm_print(2, "Vclist_assignAtoms: cell error!\n");
00314             return VRC_FAILURE;
00315         }
00316         /* Clear the counter for later use */
00317         cell->natoms = 0;
00318     }
00319
00320     /* Assign the atoms to grid points */
00321     for (iatom=0; iatom<Valist_getNumberAtoms (thee->alist); iatom++) {
00322
00323         /* Get grid span for atom */
00324         atom = Valist_getAtom(thee->alist, iatom);
00325         Vclist_gridSpan(thee, atom, imin, imax);
00326
00327         /* Now find and assign the grid points */
00328         for ( i = imin[0]; i <= imax[0]; i++) {
00329             for ( j = imin[1]; j <= imax[1]; j++) {
00330                 for ( k = imin[2]; k <= imax[2]; k++) {
00331                     /* Get index to array */
00332                     ui = Vclist_arrayIndex(thee, i, j, k);
00333                     cell = &(thee->cells[ui]);
00334                     /* Index of next available array location */
00335                     inext = cell->natoms;
00336                     cell->atoms[inext] = atom;

```

```

00337             /* Increment number of atoms */
00338             (cell->natoms)++;
00339         }
00340     }
00341   }
00342 }
00343
00344   return VRC_SUCCESS;
00345 }
00346
00347 /* Main (FORTRAN stub) constructor */
00348 VPUBLIC Vrc_Codes Vclist_ctor2(Vclist *thee, Valist *alist, double max_radius,
00349   int npts[VAPBS_DIM], Vclist_DomainMode mode,
00350   double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]) {
00351
00352   int i;
00353   VclistCell *cell;
00354
00355   /* Check and store parameters */
00356   if (Vclist_storeParms(thee, alist, max_radius, npts, mode, lower_corner,
00357     upper_corner) == VRC_FAILURE) {
00358     Vnm_print(2, "Vclist_ctor2: parameter check failed!\n");
00359     return VRC_FAILURE;
00360   }
00361
00362   /* Set up memory */
00363   thee->vmem = Vmem_ctor("APBS::VCLIST");
00364   if (thee->vmem == VNULL) {
00365     Vnm_print(2, "Vclist_ctor2: memory object setup failed!\n");
00366     return VRC_FAILURE;
00367   }
00368
00369   /* Set up cells */
00370   thee->cells = Vmem_malloc(thee->vmem, thee->n, sizeof(VclistCell));
00371   if (thee->cells == VNULL) {
00372     Vnm_print(2,
00373       "Vclist_ctor2: Failed allocating %d VclistCell objects!\n",
00374       thee->n);
00375     return VRC_FAILURE;
00376   }
00377   for (i=0; i<thee->n; i++) {
00378     cell = &(thee->cells[i]);
00379     cell->natoms = 0;
00380   }
00381
00382   /* Set up the grid */
00383   if (Vclist_setupGrid(thee) == VRC_FAILURE) {
00384     Vnm_print(2, "Vclist_ctor2: grid setup failed!\n");
00385     return VRC_FAILURE;
00386   }
00387
00388   /* Assign atoms to grid cells */
00389   if (Vclist_assignAtoms(thee) == VRC_FAILURE) {
00390     Vnm_print(2, "Vclist_ctor2: atom assignment failed!\n");
00391     return VRC_FAILURE;
00392   }
00393

```

```

00394
00395
00396
00397
00398     return VRC_SUCCESS;
00399 }
00400
00401 /* Destructor */
00402 VPUBLIC void Vclist_dtor(Vclist **thee) {
00403
00404     if ((*thee) != VNULL) {
00405         Vclist_dtor2(*thee);
00406         Vmem_free(VNULL, 1, sizeof(Vclist), (void **)thee);
00407         (*thee) = VNULL;
00408     }
00409 }
00410 }
00411
00412 /* Main (stub) destructor */
00413 VPUBLIC void Vclist_dtor2(Vclist *thee) {
00414
00415     VclistCell *cell;
00416     int i;
00417
00418     for (i=0; i<thee->n; i++) {
00419         cell = &(thee->cells[i]);
00420         VclistCell_dtor2(cell);
00421     }
00422     Vmem_free(thee->vmem, thee->n, sizeof(VclistCell),
00423             (void **)&(thee->cells));
00424     Vmem_dtor(&(thee->vmem));
00425
00426 }
00427
00428 VPUBLIC VclistCell* Vclist_getCell(Vclist *thee, double pos[VAPBS_DIM]) {
00429
00430     int i, ic[VAPBS_DIM], ui;
00431     double c[VAPBS_DIM];
00432
00433     /* Convert to grid based coordinates */
00434     for (i=0; i<VAPBS_DIM; i++) {
00435         c[i] = pos[i] - (thee->lower_corner)[i];
00436         ic[i] = (int)(c[i]/thee->spacs[i]);
00437         if (ic[i] < 0) {
00438             /* printf("OFF LOWER CORNER!\n"); */
00439             return VNULL;
00440         } else if (ic[i] >= thee->npts[i]) {
00441             /* printf("OFF UPPER CORNER!\n"); */
00442             return VNULL;
00443         }
00444     }
00445
00446     /* Get the array index */
00447     VASSERT(VAPBS_DIM == 3);
00448     ui = Vclist_arrayIndex(thee, ic[0], ic[1], ic[2]);
00449
00450     return &(thee->cells[ui]);

```

```

00451
00452 }
00453
00454 VPUBLIC VclistCell* VclistCell_ctor(int natoms) {
00455
00456     VclistCell *thee = VNULL;
00457
00458     /* Set up the structure */
00459     thee = Vmem_malloc(VNULL, 1, sizeof(VclistCell));
00460     VASSERT( thee != VNULL );
00461     VASSERT( VclistCell_ctor2(thee, natoms) == VRC_SUCCESS );
00462
00463     return thee;
00464 }
00465
00466 VPUBLIC Vrc_Codes VclistCell_ctor2(VclistCell *thee, int natoms) {
00467
00468     if (thee == VNULL) {
00469         Vnm_print(2, "VclistCell_ctor2: NULL thee!\n");
00470         return VRC_FAILURE;
00471     }
00472
00473     thee->natoms = natoms;
00474     if (thee->natoms > 0) {
00475         thee->atoms = Vmem_malloc(VNULL, natoms, sizeof(Vatom *));
00476         if (thee->atoms == VNULL) {
00477             Vnm_print(2,
00478                 "VclistCell_ctor2: unable to allocate space for %d atom pointers!\n",
00479                 natoms);
00480             return VRC_FAILURE;
00481         }
00482     }
00483
00484     return VRC_SUCCESS;
00485
00486 }
00487
00488 VPUBLIC void VclistCell_dtor(VclistCell **thee) {
00489
00490     if ((*thee) != VNULL) {
00491         VclistCell_dtor2(*thee);
00492         Vmem_free(VNULL, 1, sizeof(VclistCell), (void **)thee);
00493         (*thee) = VNULL;
00494     }
00495
00496 }
00497
00498 /* Main (stub) destructor */
00499 VPUBLIC void VclistCell_dtor2(VclistCell *thee) {
00500
00501     if (thee->natoms > 0) {
00502         Vmem_free(VNULL, thee->natoms, sizeof(Vatom *),
00503                 (void **)(&(thee->atoms)));
00504     }
00505
00506 }
00507

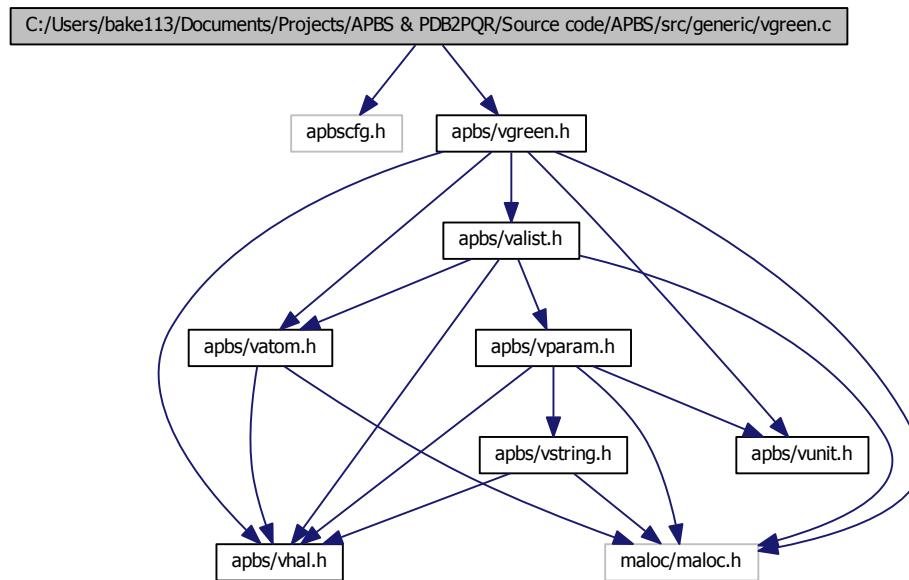
```

10.71 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vgreen.c File Reference

Class Vgreen methods.

```
#include "apbscfg.h"
#include "apbs/vgreen.h"

Include dependency graph for vgreen.c:
```



Functions

- VPRIVATE int **treesetup** (**Vgreen** *thee)
- VPRIVATE int **treetcleanup** (**Vgreen** *thee)
- VPRIVATE int **treecalc** (**Vgreen** *thee, double *xtar, double *ytar, double *ztar, double *qtar, int numtars, double *tpengtar, double *x, double *y, double *z, double *q, int numpars, double *fx, double *fy, double * fz, int iflag, int farrdim, int arrdim)
- VPUBLIC **Valist** * **Vgreen_getValist** (**Vgreen** *thee)

- Get the atom list associated with this Green's function object.
- VPUBLIC unsigned long int `Vgreen_memChk (Vgreen *thee)`
Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC `Vgreen * Vgreen_ctor (Valist *alist)`
Construct the Green's function oracle.
- VPUBLIC int `Vgreen_ctor2 (Vgreen *thee, Valist *alist)`
FORTRAN stub to construct the Green's function oracle.
- VPUBLIC void `Vgreen_dtor (Vgreen **thee)`
Destruct the Green's function oracle.
- VPUBLIC void `Vgreen_dtor2 (Vgreen *thee)`
FORTRAN stub to destruct the Green's function oracle.
- VPUBLIC int `Vgreen_helmholtz (Vgreen *thee, int npos, double *x, double *y, double *z, double *val, double kappa)`
Get the Green's function for Helmholtz's equation integrated over the atomic point charges.
- VPUBLIC int `Vgreen_helmholtzD (Vgreen *thee, int npos, double *x, double *y, double *z, double *gradx, double *grady, double *gradz, double kappa)`
Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.
- VPUBLIC int `Vgreen_coulomb_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)`
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VPUBLIC int `Vgreen_coulomb (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)`
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)
- VPUBLIC int `Vgreen_coulombD_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)`
Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VPUBLIC int `Vgreen_coulombD (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)`
Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. Johnston/R. Krasny FMM library (if available)

10.71.1 Detailed Description

Class `Vgreen` methods.

Author

Nathan Baker

Version

Id:

[vgreen.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
```

```
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vgreen.c](#).

10.72 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vgreen.c

```
00001
00057 #include "apbscfg.h"
00058 #include "apbs/vgreen.h"
00059
00060 /* Define wrappers for F77 treecode routines */
00061 #ifdef HAVE_TREE
00062 # define F77TREEPEFORCE VF77_MANGLE(treepeforce, TREEPEFORCE)
00063 # define F77DIRECT_ENG_FORCE VF77_MANGLE(direct_eng_force, DIRECT_ENG_FORCE)
00064 # define F77CLEANUP VF77_MANGLE(mycleanup, MYCLEANUP)
00065 # define F77TREE_COMPP VF77_MANGLE(mytree_compp, MYTREE_COMP)
00066 # define F77TREE_COMPPFP VF77_MANGLE(mytree_comppfp, MYTREE_COMPFP)
00067 # define F77CREATE_TREE VF77_MANGLE(mycreate_tree, MYCREATE_TREE)
00068 # define F77INITLEVELS VF77_MANGLE(myinitlevels, MYINITLEVELS)
00069 # define F77SETUP VF77_MANGLE(mysetup, MYSETUP)
00070 #endif /* ifdef HAVE_TREE */
00071
00072 /* Some constants associated with the tree code */
00073 #ifdef HAVE_TREE
00074
00075 # define FMM_DIST_TOL VSMALL
00076
00077 # define FMM_IFLAG 2
00078
00079 # define FMM_ORDER 4
00080
00081 # define FMM_THETA 0.5
00082
00083 # define FMM_MAXPARNODE 150
00084
00085 # define FMM_SHRINK 1
00086
00087 # define FMM_MINLEVEL 50000
00088
00089 # define FMM_MAXLEVEL 0
00090 #endif /* ifdef HAVE_TREE */
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108 #endif /* ifdef HAVE_TREE */
00109
00110
00111 /*
00112 * @brief Setup treecode internal structures
00113 * @ingroup Vgreen
00114 * @author Nathan Baker
00115 * @param thee Vgreen object
00116 * @return 1 if successful, 0 otherwise
00117 */
```

```

00118 VPRIIVATE int treesetup(Vgreen *thee);
00119 /*
00120  * @brief Clean up treecode internal structures
00121  * @ingroup Vgreen
00122  * @author Nathan Baker
00123  * @param thee Vgreen object
00124  * @return 1 if successful, 0 otherwise
00125  */
00127 VPRIIVATE int treecleanup(Vgreen *thee);
00128 /*
00129  * @brief Calculate forces or potential
00130  * @ingroup Vgreen
00131  * @author Nathan Baker
00132  * @param thee Vgreen object
00133  * @return 1 if successful, 0 otherwise
00134  */
00136 VPRIIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *ztar,
00137                 double *qtar, int numtars, double *tpengtar, double *x, double *y,
00138                 double *z, double *q, int numpars, double *fx, double *fy, double *fz,
00139                 int iflag, int farrdim, int arrdim);
00140
00141 #if !defined(VINLINE_VGREEN)
00142
00143 VPUBLIC Valist* Vgreen_getValist(Vgreen *thee) {
00144
00145     VASSERT(thee != VNULL);
00146     return thee->alist;
00147
00148 }
00149
00150 VPUBLIC unsigned long int Vgreen_memChk(Vgreen *thee) {
00151     if (thee == VNULL) return 0;
00152     return Vmem_bytes(thee->vmem);
00153 }
00154
00155 #endif /* if !defined(VINLINE_VGREEN) */
00156
00157 VPUBLIC Vgreen* Vgreen_ctor(Valist *alist) {
00158
00159     /* Set up the structure */
00160     Vgreen *thee = VNULL;
00161     thee = (Vgreen *)Vmem_malloc(VNULL, 1, sizeof(Vgreen));
00162     VASSERT( thee != VNULL);
00163     VASSERT( Vgreen_ctor2(thee, alist));
00164
00165     return thee;
00166 }
00167
00168 VPUBLIC int Vgreen_ctor2(Vgreen *thee, Valist *alist) {
00169
00170     VASSERT( thee != VNULL );
00171
00172     /* Memory management object */
00173     thee->vmem = Vmem_ctor("APBS:VGREEN");
00174

```

```

00175     /* Set up the atom list and grid manager */
00176     if (alist == VNULL) {
00177         Vnm_print(2, "Vgreen_ctor2: got null pointer to Valist object!\n");
00178     }
00179     thee->alist = alist;
00180
00181     /* Setup FMM tree (if applicable) */
00182 #ifdef HAVE_TREE
00183     if (!treesetup(thee)) {
00184         Vnm_print(2, "Vgreen_ctor2: Error setting up FMM tree!\n");
00185         return 0;
00186     }
00187 #endif /* ifdef HAVE_TREE */
00188
00189     return 1;
00190 }
00191
00192
00193 VPUBLIC void Vgreen_dtor(Vgreen **thee) {
00194     if ((*thee) != VNULL) {
00195         Vgreen_dtor2(*thee);
00196         Vmem_free(VNULL, 1, sizeof(Vgreen), (void **)thee);
00197         (*thee) = VNULL;
00198     }
00199 }
00200
00201 VPUBLIC void Vgreen_dtor2(Vgreen *thee) {
00202
00203 #ifdef HAVE_TREE
00204     treecleanup(thee);
00205 #endif
00206     Vmem_dtor(&(thee->vmem));
00207
00208 }
00209
00210 VPUBLIC int Vgreen_helmholtz(Vgreen *thee, int npos, double *x, double *y,
00211     double *z, double *val, double kappa) {
00212
00213     Vnm_print(2, "Error -- Vgreen_helmholtz not implemented yet!\n");
00214     return 0;
00215 }
00216
00217 VPUBLIC int Vgreen_helmholtzD(Vgreen *thee, int npos, double *x, double *y,
00218     double *z, double *gradx, double *grady, double *gradz, double kappa) {
00219
00220     Vnm_print(2, "Error -- Vgreen_helmholtzD not implemented yet!\n");
00221     return 0;
00222 }
00223
00224
00225 VPUBLIC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
00226     double *y, double *z, double *val) {
00227
00228     Vatom *atom;
00229     double *apos, charge, dist, dx, dy, dz, scale;
00230     double *q, qtemp, fx, fy, fz;
00231     int iatom, ipos;

```

```

00232
00233     if (thee == VNULL) {
00234         Vnm_print(2, "Vgreen_coulomb: Got NULL thee!\n");
00235         return 0;
00236     }
00237
00238     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00239
00240     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00241         atom = Valist_getAtom(thee->alist, iatom);
00242         apos = VatomGetPosition(atom);
00243         charge = Vatom_getCharge(atom);
00244         for (ipos=0; ipos<npos; ipos++) {
00245             dx = apos[0] - x[ipos];
00246             dy = apos[1] - y[ipos];
00247             dz = apos[2] - z[ipos];
00248             dist = VSQRT(VSQR(dx) + VSQR(dy) + VSQR(dz));
00249             if (dist > VSMALL) val[ipos] += (charge/dist);
00250         }
00251     }
00252
00253     scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00254     for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00255
00256     return 1;
00257 }
00258
00259 VPUBLIC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
00260     double *z, double *val) {
00261
00262     Vatom *atom;
00263     double *apos, charge, dist, dx, dy, dz, scale;
00264     double *q, qtemp, fx, fy, fz;
00265     int iatom, ipos;
00266
00267     if (thee == VNULL) {
00268         Vnm_print(2, "Vgreen_coulomb: Got NULL thee!\n");
00269         return 0;
00270     }
00271
00272     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00273
00274 #ifdef HAVE_TREE
00275
00276     /* Allocate charge array (if necessary) */
00277     if (Valist_getNumberAtoms(thee->alist) > 1) {
00278         if (npos > 1) {
00279             q = VNULL;
00280             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00281             if (q == VNULL) {
00282                 Vnm_print(2, "Vgreen_coulomb: Error allocating charge array!\n");
00283             }
00284         }
00285     } else {
00286         q = &(qtemp);
00287     }

```

```

00288     for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00289
00290     /* Calculate */
00291     treecalc(thee, x, y, z, q, npos, val, thee->xp, thee->yp, thee->zp,
00292             thee->qp, thee->np, &fx, &fy, &fz, 1, 1, thee->np);
00293 } else return Vgreen_coulomb_direct(thee, npos, x, y, z, val);
00294
00295 /* De-allocate charge array (if necessary) */
00296 if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **) &q);
00297
00298 scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00299 for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00300
00301 return 1;
00302
00303 #else /* ifdef HAVE_TREE */
00304
00305     return Vgreen_coulomb_direct(thee, npos, x, y, z, val);
00306
00307 #endif
00308
00309 }
00310
00311 VPUBLIC int Vgreen_coulombD_direct(Vgreen *thee, int npos,
00312         double *x, double *y, double *z, double *pot, double *gradx,
00313         double *grady, double *gradz) {
00314
00315     Vatom *atom;
00316     double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00317     double *q, qtemp;
00318     int iatom, ipos;
00319
00320     if (thee == VNULL) {
00321         Vnm_print(2, "Vgreen_coulombD: Got VNULL thee!\n");
00322         return 0;
00323     }
00324
00325     for (ipos=0; ipos<npos; ipos++) {
00326         pot[ipos] = 0.0;
00327         gradx[ipos] = 0.0;
00328         grady[ipos] = 0.0;
00329         gradz[ipos] = 0.0;
00330     }
00331
00332     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00333         atom = Valist_getAtom(thee->alist, iatom);
00334         apos = VatomGetPosition(atom);
00335         charge = VatomGetCharge(atom);
00336         for (ipos=0; ipos<npos; ipos++) {
00337             dx = apos[0] - x[ipos];
00338             dy = apos[1] - y[ipos];
00339             dz = apos[2] - z[ipos];
00340             dist2 = VSQR(dx) + VSQR(dy) + VSQR(dz);
00341             dist = VSQRT(dist2);
00342             if (dist > VSMALL) {
00343                 idist3 = 1.0/(dist*dist2);
00344                 gradx[ipos] -= (charge*dx*idist3);

```

```

00345             grady[ipos] -= (charge*dy*idist3);
00346             gradz[ipos] -= (charge*dz*idist3);
00347             pot[ipos] += (charge/dist);
00348         }
00349     }
00350 }
00351
00352 scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00353 for (ipos=0; ipos<npos; ipos++) {
00354     gradx[ipos] = gradx[ipos]*scale;
00355     grady[ipos] = grady[ipos]*scale;
00356     gradz[ipos] = gradz[ipos]*scale;
00357     pot[ipos] = pot[ipos]*scale;
00358 }
00359
00360 return 1;
00361 }
00362
00363 VPUBLIC int Vgreen_coulombD(Vgreen *thee, int npos, double *x,
00364                               double *z, double *pot, double *gradx, double *grady, double *gradz) {
00365
00366     Vatom *atom;
00367     double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00368     double *q, qtemp;
00369     int iatom, ipos;
00370
00371     if (thee == VNULL) {
00372         Vnm_print(2, "Vgreen_coulombD: Got VNULL thee!\n");
00373         return 0;
00374     }
00375
00376     for (ipos=0; ipos<npos; ipos++) {
00377         pot[ipos] = 0.0;
00378         gradx[ipos] = 0.0;
00379         grady[ipos] = 0.0;
00380         gradz[ipos] = 0.0;
00381     }
00382
00383 #ifdef HAVE_TREE
00384
00385     if (Valist_getNumberAtoms(thee->alist) > 1) {
00386         if (npos > 1) {
00387             q = VNULL;
00388             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00389             if (q == VNULL) {
00390                 Vnm_print(2, "Vgreen_coulomb: Error allocating charge array!\n")
00391 ;
00392             }
00393         } else {
00394             q = &(qtemp);
00395         }
00396         for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00397
00398         /* Calculate */
00399         treecalc(thee, x, y, z, q, npos, pot, thee->xp, thee->yp, thee->zp,
00400                  thee->qp, thee->np, gradx, grady, gradz, 2, npos, thee->np);

```

```

00401     /* De-allocate charge array (if necessary) */
00402     if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **) &q);
00403 } else return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
00404                                         gradx, grady, gradz);
00405
00406 scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00407 for (ipos=0; ipos<npos; ipos++) {
00408     gradx[ipos] = gradx[ipos]*scale;
00409     grady[ipos] = grady[ipos]*scale;
00410     gradz[ipos] = gradz[ipos]*scale;
00411     pot[ipos] = pot[ipos]*scale;
00412 }
00413
00414 return 1;
00415
00416 #else /* ifdef HAVE_TREE */
00417
00418     return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
00419                                     gradx, grady, gradz);
00420
00421 #endif
00422
00423
00424 }
00425
00426 VPRIVATE int treesetup(Vgreen *thee) {
00427
00428 #ifdef HAVE_TREE
00429
00430     double dist_tol = FMM_DIST_TOL;
00431     int iflag = FMM_IFLAG;
00432     double order = FMM_ORDER;
00433     int theta = FMM_THETA;
00434     int shrink = FMM_SHRINK;
00435     int maxparnode = FMM_MAXPARNODE;
00436     int minlevel = FMM_MINLEVEL;
00437     int maxlevel = FMM_MAXLEVEL;
00438     int level = 0;
00439     int one = 1;
00440     Vatom *atom;
00441     double xyzminmax[6], *pos;
00442     int i;
00443
00444     /* Set up particle arrays with atomic coordinates and charges */
00445     Vnm_print(0, "treesetup: Initializing FMM particle arrays...\n");
00446     thee->np = Valist_getNumberAtoms(thee->alist);
00447     thee->xp = VNULL;
00448     thee->xp = (double *) Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00449     if (thee->xp == VNULL) {
00450         Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00451                   thee->np);
00452         return 0;
00453     }
00454     thee->yp = VNULL;
00455     thee->yp = (double *) Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00456     if (thee->yp == VNULL) {
00457         Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",

```

```

00458         thee->np);
00459     return 0;
00460 }
00461 thee->zp = VNULL;
00462 thee->zp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00463 if (thee->zp == VNULL) {
00464     Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00465     thee->np);
00466     return 0;
00467 }
00468 thee->qp = VNULL;
00469 thee->qp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00470 if (thee->qp == VNULL) {
00471     Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00472     thee->np);
00473     return 0;
00474 }
00475 for (i=0; i<thee->np; i++) {
00476     atom = Valist_getAtom(thee->alist, i);
00477     pos = Vatom_getPosition(atom);
00478     thee->xp[i] = pos[0];
00479     thee->yp[i] = pos[1];
00480     thee->zp[i] = pos[2];
00481     thee->qp[i] = Vatom_getCharge(atom);
00482 }
00483
00484 Vnm_print(0, "treesetup: Setting things up...\n");
00485 F77SETUP(thee->xp, thee->yp, thee->zp, &(thee->np), &order, &theta, &iflag,
00486             &dist_tol, xyzminmax, &(thee->np));
00487
00488
00489 Vnm_print(0, "treesetup: Initializing levels...\n");
00490 F77INITLEVELS(&minlevel, &maxlevel);
00491
00492 Vnm_print(0, "treesetup: Creating tree...\n");
00493 F77CREATE_TREE(&one, &(thee->np), thee->xp, thee->yp, thee->zp, thee->qp,
00494             &shrink, &maxparnode, xyzminmax, &level, &(thee->np));
00495
00496 return 1;
00497
00498 #else /* ifdef HAVE_TREE */
00499
00500     Vnm_print(2, "treesetup: Error! APBS not linked with treecode!\n");
00501     return 0;
00502
00503 #endif /* ifdef HAVE_TREE */
00504 }
00505
00506 VPRIVATE int treecleanup(Vgreen *thee) {
00507
00508 #ifdef HAVE_TREE
00509
00510     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->xp));
00511     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->yp));
00512     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->zp));
00513     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->qp));
00514     F77CLEANUP();

```

```

00515
00516     return 1;
00517
00518 #else /* ifdef HAVE_TREE */
00519
00520     Vnm_print(2, "treecleanup: Error! APBS not linked with treecode!\n");
00521     return 0;
00522
00523 #endif /* ifdef HAVE_TREE */
00524 }
00525
00526 VPRIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *ztar,
00527                         double *qtar, int numtars, double *tpengtar, double *x, double *y,
00528                         double *z, double *q, int numpars, double *fx, double *fy, double * fz,
00529                         int iflag, int farrdim, int arrdim) {
00530
00531 #ifdef HAVE_TREE
00532     int i, level, err, maxlevel, minlevel, one;
00533     double xyzminmax[6];
00534
00535
00536     if (iflag != 1) {
00537         F77TREE_COMPFP(xtar, ytar, ztar, qtar, &numtars, tpengtar, x, y, z, q,
00538                         fx, fy, fz, &numpars, &farrdim, &arrdim);
00539     } else {
00540         F77TREE_COMPP(xtar, ytar, ztar, qtar, &numtars, tpengtar, &farrdim, x,
00541                         y, z, q, &numpars, &arrdim);
00542     }
00543
00544
00545     return 1;
00546
00547 #else /* ifdef HAVE_TREE */
00548
00549     Vnm_print(2, "treecalc: Error! APBS not linked with treecode!\n");
00550     return 0;
00551
00552 #endif /* ifdef HAVE_TREE */
00553 }
00554

```

10.73 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vparam.c File Reference

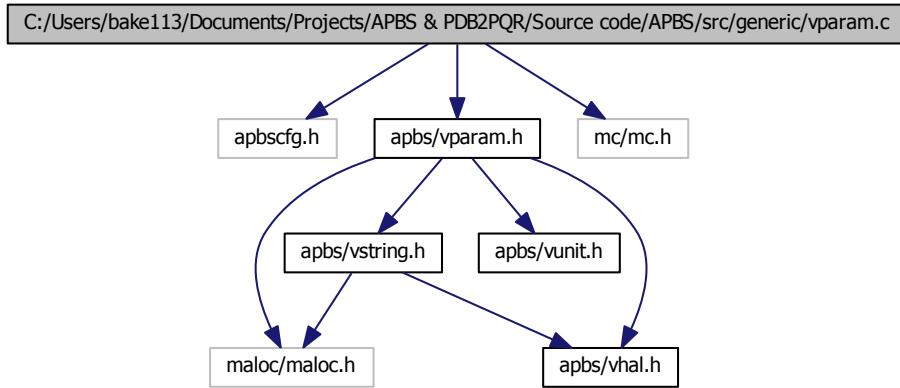
Class [Vparam](#) methods.

```

#include "apbscfg.h"
#include "apbs/vparam.h"
#include "mc/mc.h"

```

Include dependency graph for vparam.c:



Functions

- VPRIVATE int `readFlatFileLine` (Vio *sock, `Vparam_AtomData` *atom)
Read a single line of the flat file database.
- VPRIVATE int `readXMLFileAtom` (Vio *sock, `Vparam_AtomData` *atom)
Read atom information from an XML file.
- VPUBLIC unsigned long int `Vparam_memChk` (`Vparam` *thee)
Get number of bytes in this object and its members.
- VPUBLIC `Vparam_AtomData` * `Vparam_AtomData_ctor` ()
Construct the object.
- VPUBLIC int `Vparam_AtomData_ctor2` (`Vparam_AtomData` *thee)
FORTRAN stub to construct the object.
- VPUBLIC void `Vparam_AtomData_dtor` (`Vparam_AtomData` **thee)
Destroy object.
- VPUBLIC void `Vparam_AtomData_dtor2` (`Vparam_AtomData` *thee)
FORTRAN stub to destroy object.
- VPUBLIC `Vparam_ResData` * `Vparam_ResData_ctor` (`Vmem` *mem)
Construct the object.
- VPUBLIC int `Vparam_ResData_ctor2` (`Vparam_ResData` *thee, `Vmem` *mem)
FORTRAN stub to construct the object.
- VPUBLIC void `Vparam_ResData_dtor` (`Vparam_ResData` **thee)

Destroy object.

- VPUBLIC void `Vparam_ResData_dtor2` (`Vparam_ResData *thee`)
FORTRAN stub to destroy object.
- VPUBLIC `Vparam * Vparam_ctor` ()
Construct the object.
- VPUBLIC int `Vparam_ctor2` (`Vparam *thee`)
FORTRAN stub to construct the object.
- VPUBLIC void `Vparam_dtor` (`Vparam **thee`)
Destroy object.
- VPUBLIC void `Vparam_dtor2` (`Vparam *thee`)
FORTRAN stub to destroy object.
- VPUBLIC `Vparam_ResData * Vparam_getResData` (`Vparam *thee, char resName[VMAX_-ARGLEN]`)
Get residue data.
- VPUBLIC `Vparam_AtomData * Vparam_getAtomData` (`Vparam *thee, char resName[VMAX_-ARGLEN], char atomName[VMAX_ARGLEN]`)
Get atom data.
- VPUBLIC int `Vparam_readXMLFile` (`Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname`)
Read an XML format parameter database.
- VPUBLIC int `Vparam_readFlatFile` (`Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname`)
Read a flat-file format parameter database.
- VEXTERNC void `Vparam_AtomData_copyTo` (`Vparam_AtomData *thee, Vparam_AtomData *dest`)
Copy current atom object to destination.
- VEXTERNC void `Vparam_ResData_copyTo` (`Vparam_ResData *thee, Vparam_ResData *dest`)
Copy current residue object to destination.
- VEXTERNC void `Vparam_AtomData_copyFrom` (`Vparam_AtomData *thee, Vparam_AtomData *src`)
Copy current atom object from another.

Variables

- VPRIATE char * `MCwhiteChars` = " =,:;\t\n\r"
Whitespace characters for socket reads.
- VPRIATE char * `MCcommChars` = "#%"
Comment characters for socket reads.
- VPRIATE char * `MCxmlwhiteChars` = " =,:;\t\n\r<>"
Whitespace characters for XML socket reads.

10.73.1 Detailed Description

Class [Vparam](#) methods.

Author

Nathan Baker

Version

Id:

[vparam.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vparam.c](#).

10.74 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vparam.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vparam.h"
00059
00060 #if defined(HAVE_MC_H)
00061 #include "mc/mc.h"
00062 #endif
00063
00064 VEMBED(rcsid="$Id: vparam.c 1667 2011-12-02 23:22:02Z pcellis $")
00065
00066
00070 VPRIIVATE char *MCwhiteChars = " =,;\\t\\n\\r";
00071
00076 VPRIIVATE char *MCcommChars = "#%";
00077
00082 VPRIIVATE char *MCxmlwhiteChars = " =,;\\t\\n\\r<>";
00083
00092 VPRIIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData *atom);
00093
00102 VPRIIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom);
00103
00104
00105 #if !defined(VINLINE_VPARAM)
00106
00107 VPUBLIC unsigned long int Vparam_memChk(Vparam *thee) {
00108     if (thee == VNULL) return 0;
00109     return Vmem_bytes(thee->vmem);
00110 }
00111
00112 #endif /* if !defined(VINLINE_VPARAM) */
00113
00114 VPUBLIC Vparam_AtomData* Vparam_AtomData_ctor() {
00115
00116     Vparam_AtomData *thee = VNULL;
00117
00118     /* Set up the structure */
00119     thee = Vmem_malloc(VNULL, 1, sizeof(Vparam_AtomData));
00120     VASSERT(thee != VNULL);
00121     VASSERT(Vparam_AtomData_ctor2(thee));

```

```

00122     return thee;
00123 }
00125
00126 VPUBLIC int Vparam_AtomData_ctor2(Vparam_AtomData *thee) { return 1; }
00127
00128 VPUBLIC void Vparam_AtomData_dtor(Vparam_AtomData **thee) {
00129
00130     if ((*thee) != VNULL) {
00131         Vparam_AtomData_dtor2(*thee);
00132         Vmem_free(VNULL, 1, sizeof(Vparam_AtomData), (void **)thee);
00133         (*thee) = VNULL;
00134     }
00135
00136 }
00137
00138 VPUBLIC void Vparam_AtomData_dtor2(Vparam_AtomData *thee) { ; }
00139
00140 VPUBLIC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem) {
00141
00142     Vparam_ResData *thee = VNULL;
00143
00144     /* Set up the structure */
00145     thee = Vmem_malloc(mem, 1, sizeof(Vparam_ResData));
00146     VASSERT(thee != VNULL);
00147     VASSERT(Vparam_ResData_ctor2(thee, mem));
00148
00149     return thee;
00150 }
00151
00152 VPUBLIC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem) {
00153
00154     if (thee == VNULL) {
00155         Vnm_print(2, "Vparam_ResData_ctor2: Got VNULL thee!\n");
00156         return 0;
00157     }
00158     thee->vmem = mem;
00159     thee->nAtomData = 0;
00160     thee->atomData = VNULL;
00161
00162     return 1;
00163 }
00164
00165 VPUBLIC void Vparam_ResData_dtor(Vparam_ResData **thee) {
00166
00167     if ((*thee) != VNULL) {
00168         Vparam_ResData_dtor2(*thee);
00169         Vmem_free((*thee)->vmem, 1, sizeof(Vparam_ResData), (void **)thee);
00170         (*thee) = VNULL;
00171     }
00172
00173 }
00174
00175 VPUBLIC void Vparam_ResData_dtor2(Vparam_ResData *thee) {
00176
00177     if (thee == VNULL) return;
00178     if (thee->nAtomData > 0) {

```

```

00179         Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData),
00180             (void **)&(thee->atomData));
00181     }
00182     thee->nAtomData = 0;
00183     thee->atomData = VNULL;
00184 }
00185
00186 VPUBLIC Vparam* Vparam_ctor() {
00187
00188     Vparam *thee = VNULL;
00189
00190     /* Set up the structure */
00191     thee = Vmem_malloc(VNULL, 1, sizeof(Vparam) );
00192     VASSERT(thee != VNULL);
00193     VASSERT(Vparam_ctor2(thee));
00194
00195     return thee;
00196 }
00197
00198 VPUBLIC int Vparam_ctor2(Vparam *thee) {
00199
00200     if (thee == VNULL) {
00201         Vnm_print(2, "Vparam_ctor2: got VNULL thee!\n");
00202         return 0;
00203     }
00204
00205     thee->vmem = VNULL;
00206     thee->vmem = Vmem_ctor("APBS:VPARAM");
00207     if (thee->vmem == VNULL) {
00208         Vnm_print(2, "Vparam_ctor2: failed to init Vmem!\n");
00209         return 0;
00210     }
00211
00212     thee->nResData = 0;
00213     thee->resData = VNULL;
00214
00215     return 1;
00216 }
00217
00218 VPUBLIC void Vparam_dtor(Vparam **thee) {
00219
00220     if ((*thee) != VNULL) {
00221         Vparam_dtor2(*thee);
00222         Vmem_free(VNULL, 1, sizeof(Vparam), (void **)thee);
00223         (*thee) = VNULL;
00224     }
00225
00226 }
00227
00228 VPUBLIC void Vparam_dtor2(Vparam *thee) {
00229
00230     int i;
00231
00232     if (thee == VNULL) return;
00233
00234     /* Destroy the residue data */
00235     for (i=0; i<thee->nResData; i++) Vparam_ResData_dtor2(&(thee->resData[i]));

```

```

00236     if (thee->nResData > 0) Vmem_free(thee->vmem, thee->nResData,
00237         sizeof(Vparam_ResData), (void **)(&(thee->resData)));
00238     thee->nResData = 0;
00239     thee->resData = VNULL;
00240
00241     if (thee->vmem != VNULL) Vmem_dtor(&(thee->vmem));
00242     thee->vmem = VNULL;
00243
00244 }
00245
00246 VPUBLIC Vparam_ResData* Vparam_getResData(Vparam *thee,
00247     char resName[VMAX_ARGLEN]) {
00248
00249     int i;
00250     Vparam_ResData *res = VNULL;
00251
00252     VASSERT(thee != VNULL);
00253
00254     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00255         res = VNULL;
00256         return res;
00257     }
00258
00259     /* Look for the matching residue */
00260     for (i=0; i<thee->nResData; i++) {
00261         res = &(thee->resData[i]);
00262         if (Vstring_strcasecmp(resName, res->name) == 0) return res;
00263     }
00264
00265     /* Didn't find a matching residue */
00266     res = VNULL;
00267     Vnm_print(2, "Vparam_getResData: unable to find res=%s\n", resName);
00268     return res;
00269 }
00270
00271
00272 VPUBLIC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
00273     char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]) {
00274
00275     int i;
00276     Vparam_ResData *res = VNULL;
00277     Vparam_AtomData *atom = VNULL;
00278
00279     VASSERT(thee != VNULL);
00280
00281     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00282         atom = VNULL;
00283         return atom;
00284     }
00285
00286     /* Look for the matching residue */
00287     res = Vparam_getResData(thee, resName);
00288     if (res == VNULL) {
00289         atom = VNULL;
00290         Vnm_print(2, "Vparam_getAtomData: Unable to find residue %s!\n", resName);
00291         return atom;
00292     }

```

```

00293     for (i=0; i<res->nAtomData; i++) {
00294         atom = &(res->atomData[i]);
00295         if (atom == VNULL) {
00296             Vnm_print(2, "Vparam_getAtomData: got NULL atom!\n");
00297             return VNULL;
00298         }
00299         if (Vstring_strcasecmp(atomName, atom->atomName) == 0) {
00300             return atom;
00301         }
00302     }
00303     /* Didn't find a matching atom/residue */
00304     atom = VNULL;
00305     Vnm_print(2, "Vparam_getAtomData: unable to find atom '%s', res '%s'\n",
00306               atomName, resName);
00307     return atom;
00308 }
00309
00310
00311 VPUBLIC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
00312 const char *iofmt, const char *thost, const char *fname) {
00313
00314     int i, ires, natoms, nalloc, ralloc;
00315     Vparam_AtomData *atoms = VNULL;
00316     Vparam_AtomData *tatoms = VNULL;
00317     Vparam_AtomData *atom = VNULL;
00318     Vparam_ResData *res = VNULL;
00319     Vparam_ResData *residues = VNULL;
00320     Vparam_ResData *tresidues = VNULL;
00321     Vio *sock = VNULL;
00322     char currResName[VMAX_ARGLEN];
00323     char tok[VMAX_ARGLEN];
00324     char endtag[VMAX_ARGLEN];
00325
00326     VASSERT(thee != VNULL);
00327
00328     /* Setup communication */
00329     sock = Vio_ctor(iodev, iofmt, thost, fname, "r");
00330     if (sock == VNULL) {
00331         Vnm_print(2, "Vparam_readXMLFile: Problem opening virtual socket %s\n",
00332                   fname);
00333         return 0;
00334     }
00335     if (Vio_accept(sock, 0) < 0) {
00336         Vnm_print(2, "Vparam_readXMLFile: Problem accepting virtual socket %s\n",
00337                   fname);
00338         return 0;
00339     }
00340     Vio_setWhiteChars(sock, MCxmlwhiteChars);
00341     Vio_setCommChars(sock, MCcommChars);
00342
00343     /* Clear existing parameters */
00344     if (thee->nResData > 0) {
00345         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00346         for (i=0; i<thee->nResData; i++) {
00347             Vparam_ResData_dtor2(&(thee->resData[i]));
00348         }

```

```

00349     Vmem_free(thee->vmem, thee->nResData,
00350             sizeof(Vparam_ResData), (void **)(&(thee->resData)));
00351 }
00352
00353     strcpy(endtag, "/");
00354
00355 /* Set up temporary residue list */
00356
00357     ralloc = 50;
00358     residues = Vmem_malloc(thee->vmem, ralloc, sizeof(Vparam_ResData));
00359
00360 /* Read until we run out of entries, allocating space as needed */
00361 while (1) {
00362
00363     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00364
00365 /* The first token should be the start tag */
00366
00367     if (Vstring_strcasecmp(endtag, "/") == 0) strcat(endtag, tok);
00368
00369     if (Vstring_strcasecmp(tok, "residue") == 0) {
00370         if (thee->nResData >= ralloc) {
00371             residues = Vmem_malloc(thee->vmem, 2*ralloc, sizeof(
00372 Vparam_ResData));
00373             VASSERT(residues != VNULL);
00374             for (i=0; i<thee->nResData; i++) {
00375                 Vparam_ResData_copyTo(&(residues[i]), &(tresidues[i]));
00376             }
00377             Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData),
00378                     (void **)&(residues));
00379             residues = tresidues;
00380             tresidues = VNULL;
00381             ralloc = 2*ralloc;
00382         }
00383
00384 /* Initial space for this residue's atoms */
00385     nalloc = 20;
00386     natoms = 0;
00387     atoms = Vmem_malloc(thee->vmem, nalloc, sizeof(Vparam_AtomData));
00388
00389     } else if (Vstring_strcasecmp(tok, "name") == 0) {
00390         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1); /* value */
00391         strcpy(currResName, tok);
00392         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1); /* </name> */
00393     } else if (Vstring_strcasecmp(tok, "atom") == 0) {
00394         if (natoms >= nalloc) {
00395             tatoms = Vmem_malloc(thee->vmem, 2*nalloc, sizeof(
00396 Vparam_AtomData));
00397             VASSERT(tatoms != VNULL);
00398             for (i=0; i<natoms; i++) {
00399                 Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
00400             }
00401             Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),
00402                     (void **)&(atoms));
00403             atoms = tatoms;
00404             tatoms = VNULL;
00405             nalloc = 2*nalloc;

```

```

00404         }
00405         atom = &(atoms[natoms]);
00406         if (!readXMLFileAtom(sock, atom)) break;
00407         natoms++;
00408
00409     } else if (Vstring_strcasecmp(tok, "/residue") == 0) {
00410
00411     res = &(residues[thee->nResData]);
00412     Vparam_ResData_ctor2(res, thee->vmem);
00413     res->atomData = Vmem_malloc(thee->vmem, natoms,
00414                                   sizeof(Vparam_AtomData));
00415     res->nAtomData = natoms;
00416     strcpy(res->name, currResName);
00417     for (i=0; i<natoms; i++) {
00418         strcpy(atoms[i].resName, currResName);
00419         Vparam_AtomData_copyTo(&(atoms[i]), &(res->atomData[i]));
00420     }
00421     Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **) &(atoms
00422    ));
00423     (thee->nResData)++;
00424
00425     } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
00426
00427 /* Initialize and copy the residues into the Vparam object */
00428
00429     thee->resData = Vmem_malloc(thee->vmem, thee->nResData,
00430                                   sizeof(Vparam_ResData));
00431     for (ires=0; ires<thee->nResData; ires++) {
00432         Vparam_ResData_copyTo(&(residues[ires]), &(thee->resData[ires]));
00433     }
00434
00435 /* Destroy temporary atom space */
00436     Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData), (void **) &(residues));
00437
00438 /* Shut down communication */
00439     Vio_acceptFree(sock);
00440     Vio_dtor(&sock);
00441
00442     return 1;
00443
00444 VERROR1:
00445     Vnm_print(2, "Vparam_readXMLFile: Got unexpected EOF reading parameter file!\\
n");
00446     return 0;
00447
00448 }
00449
00450 VPUBLIC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
00451 const char *iofmt, const char *thost, const char *fname) {
00452
00453     int i, iatom, jatom, ires, natoms, nalloc;
00454     Vparam_AtomData *atoms = VNULL;
00455     Vparam_AtomData *tatoms = VNULL;
00456     Vparam_AtomData *atom = VNULL;
00457     Vparam_ResData *res = VNULL;
00458     Vio *sock = VNULL;

```

```

00459     char currResName[VMAX_ARGLEN];
00460
00461     VASSERT(thee != VNULL);
00462
00463     /* Setup communication */
00464     sock = Vio_ctor(iodev, iofmt, thost, fname, "r");
00465     if (sock == VNULL) {
00466         Vnm_print(2, "Vparam_readFlatFile: Problem opening virtual socket %s\n",
00467                 fname);
00468         return 0;
00469     }
00470     if (Vio_accept(sock, 0) < 0) {
00471         Vnm_print(2, "Vparam_readFlatFile: Problem accepting virtual socket %s\n"
00472                 fname);
00473         return 0;
00474     }
00475     Vio_setWhiteChars(sock, MCwhiteChars);
00476     Vio_setCommChars(sock, MCcommChars);
00477
00478     /* Clear existing parameters */
00479     if (thee->nResData > 0) {
00480         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00481         for (i=0; i<thee->nResData; i++) {
00482             Vparam_ResData_dtor2(&(thee->resData[i]));
00483         }
00484         Vmem_free(thee->vmem, thee->nResData,
00485                   sizeof(Vparam_ResData), (void **) &(thee->resData));
00486     }
00487
00488     /* Initial space for atoms */
00489     nalloc = 200;
00490     natoms = 0;
00491     atoms = Vmem_malloc(thee->vmem, nalloc, sizeof(Vparam_AtomData));
00492
00493     /* Read until we run out of entries, allocating space as needed */
00494     while (1) {
00495         if (natoms >= nalloc) {
00496             tatoms = Vmem_malloc(thee->vmem, 2*nalloc, sizeof(Vparam_AtomData));
00497             VASSERT(tatoms != VNULL);
00498             for (i=0; i<natoms; i++) {
00499                 Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
00500             }
00501             Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),
00502                       (void **) &(atoms));
00503             atoms = tatoms;
00504             tatoms = VNULL;
00505             nalloc = 2*nalloc;
00506         }
00507         atom = &(atoms[natoms]);
00508         if (!readFlatFileLine(sock, atom)) break;
00509         natoms++;
00510     }
00511     if (natoms == 0) return 0;
00512
00513     /* Count the number of residues */
00514     thee->nResData = 1;

```

```

00515     strcpy(currResName, atoms[0].resName);
00516     for (i=1; i<natoms; i++) {
00517         if (Vstring_strcasecmp(atoms[i].resName, currResName) != 0) {
00518             strcpy(currResName, atoms[i].resName);
00519             (thee->nResData)++;
00520         }
00521     }
00522
00523     /* Create the residues */
00524     thee->resData = Vmem_malloc(thee->vmem, thee->nResData,
00525         sizeof(Vparam_ResData));
00526     VASSERT(thee->resData != VNULL);
00527     for (i=0; i<(thee->nResData); i++) {
00528         res = &(thee->resData[i]);
00529         Vparam_ResData_ctor2(res, thee->vmem);
00530     }
00531
00532     /* Count the number of atoms per residue */
00533     ires = 0;
00534     res = &(thee->resData[ires]);
00535     res->nAtomData = 1;
00536     strcpy(res->name, atoms[0].resName);
00537     for (i=1; i<natoms; i++) {
00538         if (Vstring_strcasecmp(atoms[i].resName, res->name) != 0) {
00539             (ires)++;
00540             res = &(thee->resData[ires]);
00541             res->nAtomData = 1;
00542             strcpy(res->name, atoms[i].resName);
00543         } else (res->nAtomData)++;
00544     }
00545
00546     /* Allocate per-residue space for atoms */
00547     for (ires=0; ires<thee->nResData; ires++) {
00548         res = &(thee->resData[ires]);
00549         res->atomData = Vmem_malloc(thee->vmem, res->nAtomData,
00550             sizeof(Vparam_AtomData));
00551     }
00552
00553     /* Copy atoms into residues */
00554     iatom = 0;
00555     Vparam_AtomData_copyTo(&(atoms[0]), &(res->atomData[iatom]));
00556     for (ires=0; ires<thee->nResData; ires++) {
00557         res = &(thee->resData[ires]);
00558         for (jatom=0; jatom<res->nAtomData; jatom++) {
00559             Vparam_AtomData_copyTo(&(atoms[iatom]), &(res->atomData[jatom]));
00560             iatom++;
00561         }
00562     }
00563
00564     /* Shut down communication */
00565     Vio_acceptFree(sock);
00566     Vio_dtor(&sock);
00567
00568     /* Destroy temporary atom space */
00569     Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **) &(atoms));
00570
00571

```

```

00572     return 1;
00573
00574 }
00575
00576 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
00577     Vparam_AtomData *dest) {
00578
00579     VASSERT(thee != VNULL);
00580     VASSERT(dest != VNULL);
00581
00582     strcpy(dest->atomName, thee->atomName);
00583     strcpy(dest->resName, thee->resName);
00584     dest->charge = thee->charge;
00585     dest->radius = thee->radius;
00586     dest->epsilon = thee->epsilon;
00587
00588 }
00589
00590 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
00591     Vparam_ResData *dest) {
00592
00593     int i;
00594
00595     VASSERT(thee != VNULL);
00596     VASSERT(dest != VNULL);
00597
00598     strcpy(dest->name, thee->name);
00599     dest->vmem = thee->vmem;
00600     dest->nAtomData = thee->nAtomData;
00601
00602     dest->atomData = Vmem_malloc(thee->vmem, dest->nAtomData,
00603                                     sizeof(Vparam_AtomData));
00604
00605     for (i=0; i<dest->nAtomData; i++) {
00606         Vparam_AtomData_copyTo(&(thee->atomData[i]), &(dest->atomData[i]));
00607     }
00608     Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData),
00609               (void **) &(thee->atomData));
00610 }
00611
00612 VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
00613     Vparam_AtomData *src) { Vparam_AtomData_copyTo(src, thee); }
00614
00615 VPRIIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom) {
00616
00617     double dtmp;
00618     char tok[VMAX_BUFSIZE];
00619     int chgflag, radflag, nameflag;
00620
00621     VASSERT(atom != VNULL);
00622
00623     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00624
00625     chgflag = 0;
00626     radflag = 0;
00627     nameflag = 0;
00628

```

```

00629     while (1)
00630     {
00631         if (Vstring_strcasecmp(tok, "name") == 0) {
00632             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00633             if (strlen(tok) > VMAX_ARGLEN) {
00634                 Vnm_print(2, "Vparam_readXMLFileAtom: string (%s) too long \
00635 (%d)!\\n", tok, strlen(tok));
00636                 return 0;
00637             }
00638             nameflag = 1;
00639             strcpy(atom->atomName, tok);
00640         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00641             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00642             if (sscanf(tok, "%lf", &dtmp) != 1) {
00643                 Vnm_print(2, "Vparam_readXMLFileAtom: Unexpected token (%s) wh
ile \
00644 parsing charge!\\n", tok);
00645                 return 0;
00646             }
00647             chgflag = 1;
00648             atom->charge = dtmp;
00649         } else if (Vstring_strcasecmp(tok, "radius") == 0) {
00650             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00651             if (sscanf(tok, "%lf", &dtmp) != 1) {
00652                 Vnm_print(2, "Vparam_readXMLFileAtom: Unexpected token (%s) wh
ile \
00653 parsing radius!\\n", tok);
00654                 return 0;
00655             }
00656             radflag = 1;
00657             atom->radius = dtmp;
00658         } else if (Vstring_strcasecmp(tok, "epsilon") == 0) {
00659             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00660             if (sscanf(tok, "%lf", &dtmp) != 1) {
00661                 Vnm_print(2, "Vparam_readXMLFileAtom: Unexpected token (%s) wh
ile \
00662 parsing epsilon!\\n", tok);
00663                 return 0;
00664             }
00665             atom->epsilon = dtmp;
00666         } else if ((Vstring_strcasecmp(tok, "/atom") == 0) ||
00667                  (Vstring_strcasecmp(tok, "atom") == 0)) {
00668             if (chgflag && radflag && nameflag) return 1;
00669             else if (!chgflag) {
00670                 Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom witho
ut \
00671 setting the charge!\\n");
00672                 return 0;
00673             } else if (!radflag) {
00674                 Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom witho
ut \
00675 setting the radius!\\n");
00676                 return 0;
00677             } else if (!nameflag) {
00678                 Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom witho
ut \
00679 setting the name!\\n");

```

```

00680             return 0;
00681         }
00682     }
00683     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00684 }
00685
00686 /* If we get here something wrong has happened */
00687
00688 VJMPERR1(1);
00689
00690 VERROR1:
00691     Vnm_print(2, "Vparam_readXMLFileAtom: Got unexpected EOF reading parameter fi
le!\\n");
00692     return 0;
00693
00694 }
00695
00696 VPRIIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData *atom) {
00697
00698     double dtmp;
00699     char tok[VMAX_BUFSIZE];
00700
00701     VASSERT(atom != VNULL);
00702
00703     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00704     if (strlen(tok) > VMAX_ARGLEN) {
00705         Vnm_print(2, "Vparam_readFlatFile: string (%s) too long (%d)!\\n",
00706                 tok, strlen(tok));
00707         return 0;
00708     }
00709     strcpy(atom->resName, tok);
00710     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00711     if (strlen(tok) > VMAX_ARGLEN) {
00712         Vnm_print(2, "Vparam_readFlatFile: string (%s) too long (%d)!\\n",
00713                 tok, strlen(tok));
00714         return 0;
00715     }
00716     strcpy(atom->atomName, tok);
00717     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00718     if (sscanf(tok, "%lf", &dtmp) != 1) {
00719         Vnm_print(2, "Vparam_readFlatFile: Unexpected token (%s) while \
00720 parsing charge!\\n", tok);
00721         return 0;
00722     }
00723     atom->charge = dtmp;
00724     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00725     if (sscanf(tok, "%lf", &dtmp) != 1) {
00726         Vnm_print(2, "Vparam_readFlatFile: Unexpected token (%s) while \
00727 parsing radius!\\n", tok);
00728         return 0;
00729     }
00730     atom->radius = dtmp;
00731     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00732     if (sscanf(tok, "%lf", &dtmp) != 1) {
00733         Vnm_print(2, "Vparam_readFlatFile: Unexpected token (%s) while \
00734 parsing radius!\\n", tok);
00735         return 0;

```

```

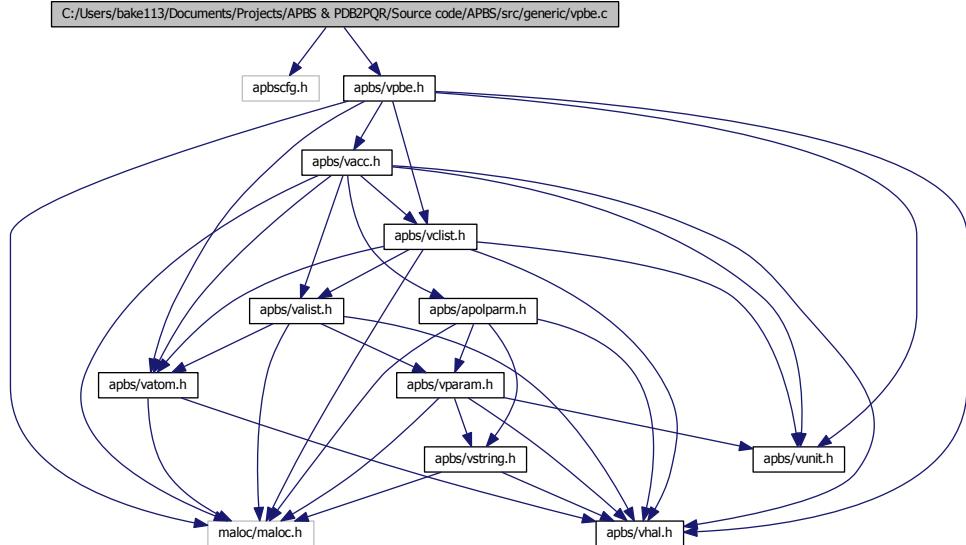
00736     }
00737     atom->epsilon = dtmp;
00738
00739     return 1;
00740
00741 VERROR1:
00742     Vnm_print(2, "Vparam_readFlatFile: Got unexpected EOF reading parameter file!
00743 \n");
00743     return 0;
00744 }
```

10.75 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vpbe.c File Reference

Class Vpbe methods.

```
#include "apbscfg.h"
#include "apbs/vpbe.h"
```

Include dependency graph for vpbe.c:



Defines

- #define **MAX_SPLINE_WINDOW** 0.5

Functions

- VPUBLIC Valist * Vpbe_getValist (Vpbe *thee)
Get atom list.
- VPUBLIC Vacc * Vpbe_getVacc (Vpbe *thee)
Get accessibility oracle.
- VPUBLIC double Vpbe_getBulkIonicStrength (Vpbe *thee)
Get bulk ionic strength.
- VPUBLIC double Vpbe_getTemperature (Vpbe *thee)
Get temperature.
- VPUBLIC double Vpbe_getSoluteDiel (Vpbe *thee)
Get solute dielectric constant.
- VPUBLIC double * Vpbe_getSoluteCenter (Vpbe *thee)
Get coordinates of solute center.
- VPUBLIC double Vpbe_getSolventDiel (Vpbe *thee)
Get solvent dielectric constant.
- VPUBLIC double Vpbe_getSolventRadius (Vpbe *thee)
Get solvent molecule radius.
- VPUBLIC double Vpbe_getMaxIonRadius (Vpbe *thee)
Get maximum radius of ion species.
- VPUBLIC double Vpbe_getXkappa (Vpbe *thee)
Get Debye-Hückel parameter.
- VPUBLIC double Vpbe_getDeblen (Vpbe *thee)
Get Debye-Hückel screening length.
- VPUBLIC double Vpbe_getZkappa2 (Vpbe *thee)
Get modified squared Debye-Hückel parameter.
- VPUBLIC double Vpbe_getZmagic (Vpbe *thee)
Get charge scaling factor.
- VPUBLIC double Vpbe_getSoluteRadius (Vpbe *thee)
Get sphere radius which bounds biomolecule.
- VPUBLIC double Vpbe_getSoluteXlen (Vpbe *thee)
Get length of solute in x dimension.
- VPUBLIC double Vpbe_getSoluteYlen (Vpbe *thee)
Get length of solute in y dimension.
- VPUBLIC double Vpbe_getSoluteZlen (Vpbe *thee)
Get length of solute in z dimension.

- VPUBLIC double `Vpbe_getSoluteCharge` (`Vpbe *thee`)

Get total solute charge.
- VPUBLIC double `Vpbe_getzmem` (`Vpbe *thee`)

Get z position of the membrane bottom.
- VPUBLIC double `Vpbe_getLmem` (`Vpbe *thee`)

Get length of the membrane (A)
aauthor Michael Grabe.
- VPUBLIC double `Vpbe_getmembraneDiel` (`Vpbe *thee`)

Get membrane dielectric constant.
- VPUBLIC double `Vpbe_getmemv` (`Vpbe *thee`)

Get membrane potential (kT)
- VPUBLIC `Vpbe *` `Vpbe_ctor` (`Valist *alist`, int `ionNum`, double `*ionConc`, double `*ionRadii`, double `*ionQ`, double `T`, double `soluteDiel`, double `solventDiel`, double `solventRadius`, int `focusFlag`, double `sdens`, double `z_mem`, double `L`, double `membraneDiel`, double `V`)

Construct Vpbe object.
- VPUBLIC int `Vpbe_ctor2` (`Vpbe *thee`, `Valist *alist`, int `ionNum`, double `*ionConc`, double `*ionRadii`, double `*ionQ`, double `T`, double `soluteDiel`, double `solventDiel`, double `solventRadius`, int `focusFlag`, double `sdens`, double `z_mem`, double `L`, double `membraneDiel`, double `V`)

FORTRAN stub to construct Vpbe objct.
- VPUBLIC void `Vpbe_dtor` (`Vpbe **thee`)

Object destructor.
- VPUBLIC void `Vpbe_dtor2` (`Vpbe *thee`)

FORTRAN stub object destructor.
- VPUBLIC double `Vpbe_getCoulombEnergy1` (`Vpbe *thee`)

Calculate coulombic energy of set of charges.
- VPUBLIC unsigned long int `Vpbe_memChk` (`Vpbe *thee`)

Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC int `Vpbe_getIons` (`Vpbe *thee`, int `*nion`, double `ionConc[MAXION]`, double `ionRadii[MAXION]`, double `ionQ[MAXION]`)

Get information about the counterion species present.

10.75.1 Detailed Description

Class Vpbe methods.

Author

Nathan Baker

Version

Id:

[vpbe.c](#) 1667 2011-12-02 23:22:02Z pcells

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [vpbe.c](#).

10.76 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vpbe.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vpbe.h"
00059
00060 /* //////////////////////////////// */
00061 // Class Vpbe: Private method declaration
00063 #define MAX_SPLINE_WINDOW 0.5
00064
00065 /* //////////////////////////////// */
00066 // Class Vpbe: Inlineable methods
00068 #if !defined(VINLINE_VPBE)
00069
00070 VPUBLIC Valist* Vpbe_getValist(Vpbe *thee) {
00071     VASSERT(thee != VNULL);
00073     return thee->alist;
00074
00075 }
00076
00077 VPUBLIC Vacc* Vpbe_getVacc(Vpbe *thee) {
00078     VASSERT(thee != VNULL);
00079     VASSERT(thee->paramFlag);
00080     return thee->acc;
00082
00083 }
00084
00085 VPUBLIC double Vpbe_getBulkIonicStrength(Vpbe *thee) {
00086
00087     VASSERT(thee != VNULL);
00088     VASSERT(thee->paramFlag);
00089     return thee->bulkIonicStrength;
00090 }
00091
00092 VPUBLIC double Vpbe_getTemperature(Vpbe *thee) {
00093
00094     VASSERT(thee != VNULL);
00095     VASSERT(thee->paramFlag);
00096     return thee->T;
00097
00098 }
00099
00100 VPUBLIC double Vpbe_getSoluteDiel(Vpbe *thee) {
00101
00102     VASSERT(thee != VNULL);
00103     VASSERT(thee->paramFlag);
00104     return thee->soluteDiel;
00105
00106 }
00107
00108 VPUBLIC double* Vpbe_getSoluteCenter(Vpbe *thee) {
00109
00110     VASSERT(thee != VNULL);

```

```
00111     return thee->soluteCenter;
00112 }
00113
00114 VPUBLIC double Vpbe_getSolventDiel(Vpbe *thee) {
00115
00116     VASSERT(thee != VNULL);
00117     VASSERT(thee->paramFlag);
00118     return thee->solventDiel;
00119 }
00120
00121 VPUBLIC double Vpbe_getSolventRadius(Vpbe *thee) {
00122
00123     VASSERT(thee != VNULL);
00124     VASSERT(thee->paramFlag);
00125     return thee->solventRadius;
00126 }
00127
00128 VPUBLIC double Vpbe_getMaxIonRadius(Vpbe *thee) {
00129
00130     VASSERT(thee != VNULL);
00131     VASSERT(thee->paramFlag);
00132     return thee->maxIonRadius;
00133 }
00134
00135 VPUBLIC double Vpbe_getXkappa(Vpbe *thee) {
00136
00137     VASSERT(thee != VNULL);
00138     VASSERT(thee->paramFlag);
00139     return thee->xkappa;
00140 }
00141
00142 VPUBLIC double Vpbe_getDeblen(Vpbe *thee) {
00143
00144     VASSERT(thee != VNULL);
00145     VASSERT(thee->paramFlag);
00146     return thee->deblen;
00147 }
00148
00149 VPUBLIC double Vpbe_getZkappa2(Vpbe *thee) {
00150
00151     VASSERT(thee != VNULL);
00152     VASSERT(thee->paramFlag);
00153     return thee->z kappa2;
00154 }
00155
00156 VPUBLIC double Vpbe_getZmagic(Vpbe *thee) {
00157
00158     VASSERT(thee != VNULL);
00159     VASSERT(thee->paramFlag);
00160     return thee->zmagic;
00161 }
00162
00163 VPUBLIC double Vpbe_getSoluteRadius(Vpbe *thee) {
00164
00165     VASSERT(thee != VNULL);
00166     return thee->soluteRadius;
00167 }
```

```
00168
00169 VPUBLIC double Vpbe_getSoluteXlen(Vpbe *thee) {
00170
00171     VASSERT(thee != VNULL);
00172     return thee->soluteXlen;
00173 }
00174
00175 VPUBLIC double Vpbe_getSoluteYlen(Vpbe *thee) {
00176
00177     VASSERT(thee != VNULL);
00178     return thee->soluteYlen;
00179 }
00180
00181 VPUBLIC double Vpbe_getSoluteZlen(Vpbe *thee) {
00182
00183     VASSERT(thee != VNULL);
00184     return thee->soluteZlen;
00185 }
00186
00187 VPUBLIC double Vpbe_getSoluteCharge(Vpbe *thee) {
00188
00189     VASSERT(thee != VNULL);
00190     return thee->soluteCharge;
00191 }
00192
00193 /* /////////////////////////////////
00194 // Routine: Vpbe_getzmem
00195 // Purpose: This routine returns values stored in the structure thee.
00196 // Author: Michael Grabe
00197 VPUBLIC double Vpbe_getzmem(Vpbe *thee) {
00198
00199     VASSERT(thee != VNULL);
00200     VASSERT(thee->param2Flag);
00201     return thee->z_mem;
00202 }
00203
00204
00205 /* /////////////////////////////////
00206 // Routine: Vpbe_getLmem
00207 // Purpose: This routine returns values stored in the structure thee.
00208 // Author: Michael Grabe
00209 VPUBLIC double Vpbe_getLmem(Vpbe *thee) {
00210
00211     VASSERT(thee != VNULL);
00212     VASSERT(thee->param2Flag);
00213     return thee->L;
00214 }
00215
00216
00217 /* /////////////////////////////////
00218 // Routine: Vpbe_getmembraneDiel
00219 // Purpose: This routine returns values stored in the structure thee.
00220 // Author: Michael Grabe
00221 VPUBLIC double Vpbe_getmembraneDiel(Vpbe *thee) {
00222
00223     VASSERT(thee != VNULL);
00224     VASSERT(thee->param2Flag);
00225     return thee->mMembraneDiel;
00226 }
00227 }
```

```

00228 /* /////////////////////////////////
00229 // Routine: Vpbe_getmemv
00230 // Purpose: This routine returns values stored in the structure thee.
00231 // Author: Michael Grabe
00234 VPUBLIC double Vpbe_getmemv(Vpbe *thee) {
00235
00236 VASSERT(thee != VNULL);
00237 VASSERT(thee->param2Flag);
00238 return thee->V;
00239 }
00240
00241 #endif /* if !defined(VINLINE_VPBE) */
00242
00243 /* /////////////////////////////////
00244 // Class Vpbe: Non-inlineable methods
00245
00247 VPUBLIC Vpbe* Vpbe_ctor(Valist *alist, int ionNum, double *ionConc,
00248     double *ionRadii, double *ionQ, double T,
00249     double soluteDiel, double solventDiel,
00250     double solventRadius, int focusFlag, double sdens,
00251     double z_mem, double L, double membraneDiel, double V ) {
00252
00253 /* Set up the structure */
00254 Vpbe *thee = VNULL;
00255 thee = Vmem_malloc(VNULL, 1, sizeof(Vpbe) );
00256 VASSERT( thee != VNULL);
00257 VASSERT( Vpbe_ctor2(thee, alist, ionNum, ionConc, ionRadii, ionQ,
00258     T, soluteDiel, solventDiel, solventRadius, focusFlag, sdens,
00259     z_mem, L, membraneDiel, V) );
00260
00261 return thee;
00262 }
00263
00264
00265 VPUBLIC int Vpbe_ctor2(Vpbe *thee, Valist *alist, int ionNum,
00266     double *ionConc, double *ionRadii,
00267     double *ionQ, double T, double soluteDiel,
00268     double solventDiel, double solventRadius, int focusFlag,
00269     double sdens, double z_mem, double L, double membraneDiel,
00270     double V) {
00271
00272     int i, iatom, inhash[3];
00273     double atomRadius;
00274     Vatom *atom;
00275     double center[3] = {0.0, 0.0, 0.0};
00276     double lower_corner[3] = {0.0, 0.0, 0.0};
00277     double upper_corner[3] = {0.0, 0.0, 0.0};
00278     double disp[3], dist, radius, charge, xmin, xmax, ymin, ymax, zmin, zmax;
00279     double x, y, z, netCharge;
00280     double nhash[3];
00281     const double N_A = 6.022045000e+23;
00282     const double e_c = 4.803242384e-10;
00283     const double k_B = 1.380662000e-16;
00284     const double pi = 4. * VATAN(1.);
00285
00286 /* Set up memory management object */

```

```

00287     thee->vmem = Vmem_ctor("APBS::VPBE");
00288
00289     VASSERT(thee != VNULL);
00290     if (alist == VNULL) {
00291         Vnm_print(2, "Vpbe_ctor2: Got null pointer to Valist object!\n");
00292         return 0;
00293     }
00294
00295     /* ***** STUFF THAT GETS DONE FOR EVERYONE ***** */
00296     /* Set pointers */
00297     thee->alist = alist;
00298     thee->paramFlag = 0;
00299
00300     /* Determine solute center */
00301     center[0] = thee->alist->center[0];
00302     center[1] = thee->alist->center[1];
00303     center[2] = thee->alist->center[2];
00304     thee->soluteCenter[0] = center[0];
00305     thee->soluteCenter[1] = center[1];
00306     thee->soluteCenter[2] = center[2];
00307
00308     /* Determine solute length and charge*/
00309     radius = 0;
00310     atom = Valist_getAtom(thee->alist, 0);
00311     xmin = Vatom_getPosition(atom)[0];
00312     xmax = Vatom_getPosition(atom)[0];
00313     ymin = Vatom_getPosition(atom)[1];
00314     ymax = Vatom_getPosition(atom)[1];
00315     zmin = Vatom_getPosition(atom)[2];
00316     zmax = Vatom_getPosition(atom)[2];
00317     charge = 0;
00318     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00319         atom = Valist_getAtom(thee->alist, iatom);
00320         atomRadius = Vatom_getRadius(atom);
00321         x = Vatom_getPosition(atom)[0];
00322         y = Vatom_getPosition(atom)[1];
00323         z = Vatom_getPosition(atom)[2];
00324         if ((x+atomRadius) > xmax) xmax = x + atomRadius;
00325         if ((x-atomRadius) < xmin) xmin = x - atomRadius;
00326         if ((y+atomRadius) > ymax) ymax = y + atomRadius;
00327         if ((y-atomRadius) < ymin) ymin = y - atomRadius;
00328         if ((z+atomRadius) > zmax) zmax = z + atomRadius;
00329         if ((z-atomRadius) < zmin) zmin = z - atomRadius;
00330         disp[0] = (x - center[0]);
00331         disp[1] = (y - center[1]);
00332         disp[2] = (z - center[2]);
00333         dist = (disp[0]*disp[0]) + (disp[1]*disp[1]) + (disp[2]*disp[2]);
00334         dist = VSQRT(dist) + atomRadius;
00335         if (dist > radius) radius = dist;
00336         charge += Vatom_getCharge(Valist_getAtom(thee->alist, iatom));
00337     }
00338     thee->soluteRadius = radius;
00339     Vnm_print(0, "Vpbe_ctor2: solute radius = %g\n", radius);
00340     thee->soluteXlen = xmax - xmin;
00341     thee->soluteYlen = ymax - ymin;
00342     thee->soluteZlen = zmax - zmin;
00343     Vnm_print(0, "Vpbe_ctor2: solute dimensions = %g x %g x %g\n",

```

```

00344         thee->soluteXlen, thee->soluteYlen, thee->soluteZlen);
00345     thee->soluteCharge = charge;
00346     Vnm_print(0, "Vpbe_ctor2:  solute charge = %g\n", charge);
00347
00348     /* Set parameters */
00349     thee->numIon = ionNum;
00350     if (thee->numIon >= MAXION) {
00351         Vnm_print(2, "Vpbe_ctor2:  Too many ion species (MAX = %d) !\n",
00352                 MAXION);
00353         return 0;
00354     }
00355     thee->bulkIonicStrength = 0.0;
00356     thee->maxIonRadius = 0.0;
00357     netCharge = 0.0;
00358     for (i=0; i<thee->numIon; i++) {
00359         thee->ionConc[i] = ionConc[i];
00360         thee->ionRadii[i] = ionRadii[i];
00361         if (ionRadii[i] > thee->maxIonRadius) thee->maxIonRadius = ionRadii[i];
00362         thee->ionQ[i] = ionQ[i];
00363         thee->bulkIonicStrength += (0.5*ionConc[i]*VSQR(ionQ[i]));
00364         netCharge += (ionConc[i]*ionQ[i]);
00365     }
00366 #ifndef VAPBSQUIET
00367     Vnm_print(1, " Vpbe_ctor:  Using max ion radius (%g A) for exclusion \
00368 function\n", thee->maxIonRadius);
00369 #endif
00370     if (VABS(netCharge) > VSMALL) {
00371         Vnm_print(2, "Vpbe_ctor2:  You have a counterion charge imbalance!\n");
00372         Vnm_print(2, "Vpbe_ctor2:  Net charge conc. = %g M\n", netCharge);
00373         return 0;
00374     }
00375     thee->T = T;
00376     thee->soluteDiel = soluteDiel;
00377     thee->solventDiel = solventDiel;
00378     thee->solventRadius = solventRadius;
00379
00380     /* Compute parameters:
00381      *
00382      * kappa^2 = (8 pi N_A e_c^2) I_s / (1000 eps_w k_B T)
00383      * kappa = 0.325567 * I_s^{1/2} angstroms^{-1}
00384      * deblen = 1 / kappa
00385      *      = 3.071564378 * I_s^{1/2} angstroms
00386      * \bar{kappa}^2 = eps_w * kappa^2
00387      * zmagic = (4 * pi * e_c^2) / (k_B T)   (we scale the diagonal later)
00388      *      = 7046.528838
00389      */
00390     if (thee->T == 0.0) {
00391         Vnm_print(2, "Vpbe_ctor2:  You set the temperature to 0 K.\n");
00392         Vnm_print(2, "Vpbe_ctor2:  That violates the 3rd Law of Thermo.!");
00393         return 0;
00394     }
00395     if (thee->bulkIonicStrength == 0.) {
00396         thee->xkappa = 0.;
00397         thee->deblen = 0.;
00398         thee->zkappa2 = 0.;
00399     } else {
00400         thee->xkappa = VSQRT( thee->bulkIonicStrength * 1.0e-16 *

```

```

00401         ((8.0 * pi * N_A * e_c*e_c) /
00402          (1000.0 * thee->solventDiel * k_B * T))
00403     );
00404     thee->deblen = 1. / thee->xkappa;
00405     thee->zkappa2 = thee->solventDiel * VSQR(thee->xkappa);
00406   }
00407   Vnm_print(0, "Vpbe_ctor2: bulk ionic strength = %g\n",
00408             thee->bulkIonicStrength);
00409   Vnm_print(0, "Vpbe_ctor2: xkappa = %g\n", thee->xkappa);
00410   Vnm_print(0, "Vpbe_ctor2: Debye length = %g\n", thee->deblen);
00411   Vnm_print(0, "Vpbe_ctor2: zkappa2 = %g\n", thee->zkappa2);
00412   thee->zmagic = ((4.0 * pi * e_c*e_c) / (k_B * thee->T)) * 1.0e+8;
00413   Vnm_print(0, "Vpbe_ctor2: zmagic = %g\n", thee->zmagic);
00414
00415 /* Compute accessibility objects:
00416    * - Allow for extra room in the case of spline windowing
00417    * - Place some limits on the size of the hash table in the case of very
00418    *   large molecules
00419 */
00420   if (thee->maxIonRadius > thee->solventRadius)
00421     radius = thee->maxIonRadius + MAX_SPLINE_WINDOW;
00422   else radius = thee->solventRadius + MAX_SPLINE_WINDOW;
00423
00424   nhash[0] = (thee->soluteXlen)/0.5;
00425   nhash[1] = (thee->soluteYlen)/0.5;
00426   nhash[2] = (thee->soluteZlen)/0.5;
00427   for (i=0; i<3; i++) inhash[i] = (int)(nhash[i]);
00428
00429   for (i=0;i<3;i++){
00430     if (inhash[i] < 3) inhash[i] = 3;
00431     if (inhash[i] > MAX_HASH_DIM) inhash[i] = MAX_HASH_DIM;
00432   }
00433   Vnm_print(0, "Vpbe_ctor2: Constructing Vclist with %d x %d x %d table\n",
00434             inhash[0], inhash[1], inhash[2]);
00435
00436   thee->clist = Vclist_ctor(thee->alist, radius, inhash,
00437                             CLIST_AUTO_DOMAIN, lower_corner, upper_corner);
00438
00439   VASSERT(thee->clist != VNNULL);
00440   thee->acc = Vacc_ctor(thee->alist, thee->clist, sdens);
00441
00442   VASSERT(thee->acc != VNNULL);
00443
00444 /* SMPBE Added */
00445   thee->smsize = 0.0;
00446   thee->smvolume = 0.0;
00447   thee->ipkey = 0;
00448
00449   thee->paramFlag = 1;
00450
00451 /*-----*/
00452 /* added by Michael Grabe */ */
00453 /*-----*/
00454
00455   thee->z_mem = z_mem;
00456   thee->L = L;
00457   thee->membraneDiel = membraneDiel;

```

```

00458     thee->V = V;
00459
00460 //     if (V != VNULL) thee->param2Flag = 1;
00461 //     else thee->param2Flag = 0;
00462
00463 /*-----*/
00464
00465     return 1;
00466 }
00467
00468 VPUBLIC void Vpbe_dtor(Vpbe **thee) {
00469     if ((*thee) != VNULL) {
00470         Vpbe_dtor2(*thee);
00471         Vmem_free(VNULL, 1, sizeof(Vpbe), (void **)thee);
00472         (*thee) = VNULL;
00473     }
00474 }
00475
00476 VPUBLIC void Vpbe_dtor2(Vpbe *thee) {
00477     Vclist_dtor(&(thee->clist));
00478     Vacc_dtor(&(thee->acc));
00479     Vmem_dtor(&(thee->vmem));
00480 }
00481
00482 VPUBLIC double Vpbe_getCoulombEnergy1(Vpbe *thee) {
00483
00484     int i, j, k, natoms;
00485
00486     double dist, *ipos, *jpos, icharge, jcharge;
00487     double energy = 0.0;
00488     double eps, T;
00489     Vatom *iatom, *jatom;
00490     Valist *alist;
00491
00492     VASSERT(thee != VNULL);
00493     alist = Vpbe_getValist(thee);
00494     VASSERT(alist != VNULL);
00495     natoms = Valist_getNumberAtoms(alist);
00496
00497     /* Do the sum */
00498     for (i=0; i<natoms; i++) {
00499         iatom = Valist_getAtom(alist,i);
00500         icharge = Vatom_getCharge(iatom);
00501         ipos = Vatom_getPosition(iatom);
00502         for (j=i+1; j<natoms; j++) {
00503             jatom = Valist_getAtom(alist,j);
00504             jcharge = Vatom_getCharge(jatom);
00505             jpos = Vatom_getPosition(jatom);
00506             dist = 0;
00507             for (k=0; k<3; k++) dist += ((ipos[k]-jpos[k])*(ipos[k]-jpos[k]));
00508             dist = VSQRT(dist);
00509             energy = energy + icharge*jcharge/dist;
00510         }
00511     }
00512
00513     /* Convert the result to J */
00514     T = Vpbe_getTemperature(thee);

```

```

00515     eps = Vpbe_getSoluteDielectric(thee);
00516     energy = energy*Vunit_ec*Vunit_ec/(4*Vunit_pi*Vunit_eps0*eps*(1.0e-10));
00517
00518     /* Scale by Boltzmann energy */
00519     energy = energy/(Vunit_kb*T);
00520
00521     return energy;
00522 }
00523
00524 VPUBLIC unsigned long int Vpbe_memChk(Vpbe *thee) {
00525
00526     unsigned long int memUse = 0;
00527
00528     if (thee == VNULL) return 0;
00529
00530     memUse = memUse + sizeof(Vpbe);
00531     memUse = memUse + (unsigned long int)Vacc_memChk(thee->acc);
00532
00533     return memUse;
00534 }
00535
00536 VPUBLIC int Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[MAXION],
00537     double ionRadii[MAXION], double ionQ[MAXION]) {
00538
00539     int i;
00540
00541     VASSERT(thee != VNULL);
00542
00543     *nion = thee->numIon;
00544     for (i=0; i<(*nion); i++) {
00545         ionConc[i] = thee->ionConc[i];
00546         ionRadii[i] = thee->ionRadii[i];
00547         ionQ[i] = thee->ionQ[i];
00548     }
00549
00550     return *nion;
00551 }

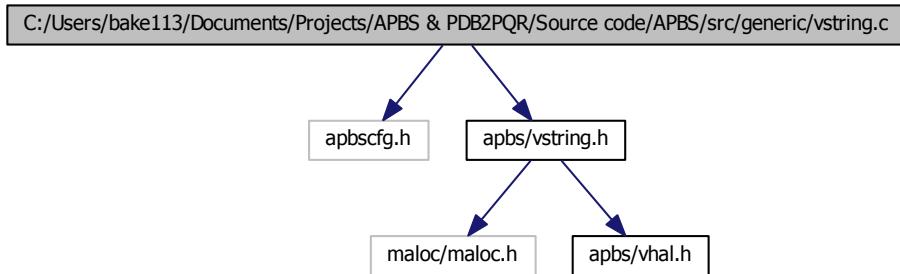
```

10.77 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vstring.c File Reference

Class Vstring methods.

```
#include "apbscfg.h"
#include "apbs/vstring.h"
```

Include dependency graph for vstring.c:



Functions

- VPUBLIC int [Vstring_strcasecmp](#) (const char *s1, const char *s2)
Case-insensitive string comparison (BSD standard)
- VPUBLIC int [Vstring_isdigit](#) (const char *tok)
A modified sscanf that examines the complete string.

10.77.1 Detailed Description

Class Vstring methods.

Author

Nathan Baker

Version

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*
```

```

* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vstring.c](#).

10.78 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/generic/vstring.c

```

00001
00056 #include "apbscfg.h"
00057 #include "apbs/vstring.h"
00058
00059 /* ///////////////////////////////// */
00060 // Routine: Vstring_strcasecmp
00061 //
00062 // Copyright (c) 1988-1993 The Regents of the University of
00063 // California.

```

```

00064 // Copyright (c) 1995-1996 Sun Microsystems, Inc.
00065 VPUBLIC int Vstring_strcasecmp(const char *s1, const char *s2) {
00067
00068 #if !defined(HAVE_STRCASECMP)
00069     unsigned char charmap[] = {
00070         0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
00071         0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
00072         0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
00073         0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
00074         0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
00075         0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
00076         0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
00077         0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
00078         0x40, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
00079         0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
00080         0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
00081         0x78, 0x79, 0x7a, 0x5b, 0x5c, 0x5d, 0x5e, 0x5f,
00082         0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
00083         0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
00084         0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
00085         0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
00086         0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
00087         0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
00088         0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
00089         0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
00090         0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
00091         0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
00092         0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
00093         0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
00094         0xc0, 0xe1, 0xe2, 0xe3, 0xe4, 0xc5, 0xe6, 0xe7,
00095         0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
00096         0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
00097         0xf8, 0xf9, 0xfa, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
00098         0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
00099         0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
00100         0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
00101         0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
00102     };
00103
00104     unsigned char u1, u2;
00105
00106     for ( ; ; s1++, s2++) {
00107         u1 = (unsigned char) *s1;
00108         u2 = (unsigned char) *s2;
00109         if ((u1 == '\0') || (charmap[u1] != charmap[u2])) {
00110             break;
00111         }
00112     }
00113     return charmap[u1] - charmap[u2];
00114
00115 #else
00116
00117     return strcasecmp(s1, s2);
00118
00119 #endif
00120
00121 }
```

```

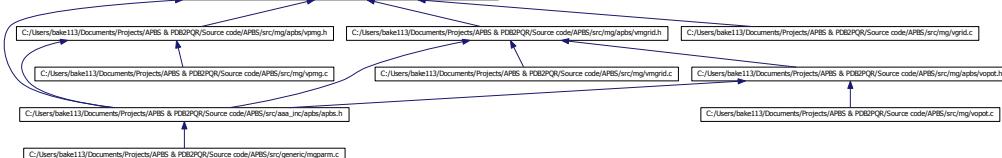
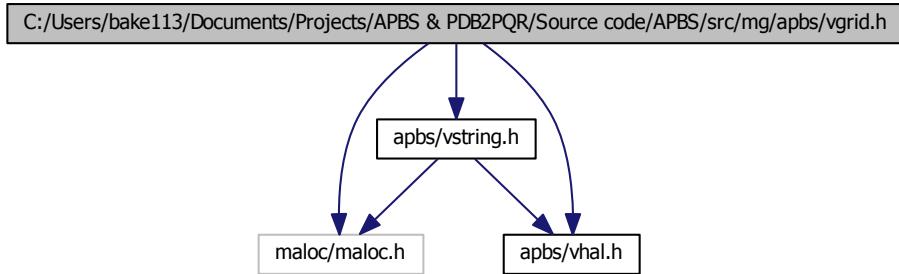
00122
00123 /* /////////////////////////////////
00124 // Routine: Vstring_isdigit
00125 //
00126 //           Improves upon sscanf to see if a token is an int or not
00127 //
00128 //           Returns isdigit: 1 if a digit, 0 otherwise
00129 VPUBLIC int Vstring_isdigit(const char *tok) {
00130     int i, isdigit, ti;
00131     char checkchar[1];
00132     char name[VMAX_BUFSIZE];
00133     strcpy(name,tok);
00134     isdigit = 1;
00135     for(i=0; ; i++){
00136         checkchar[0] = name[i];
00137         if (name[i] == '\0'){
00138             break;
00139         }
00140         if (sscanf(checkchar, "%d", &ti) != 1){
00141             isdigit = 0;
00142             break;
00143         }
00144     }
00145     return isdigit;
00146 }
00147 }
```

10.79 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vgrid.h File Reference

Potential oracle for Cartesian mesh data.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vstring.h"
```

Include dependency graph for vgrid.h:



Data Structures

- struct **sVgrid**
Electrostatic potential oracle for Cartesian mesh data.

Defines

- #define **VGRID_DIGITS** 6
Number of decimal places for comparisons and formatting.

TypeDefs

- typedef struct **sVgrid** **Vgrid**

Declaration of the Vgrid class as the `sVgrid` structure.

Functions

- VEXTERNC unsigned long int `Vgrid_memChk` (`Vgrid *thee`)

Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC `Vgrid *Vgrid_ctor` (int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)

Construct Vgrid object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int `Vgrid_ctor2` (`Vgrid *thee`, int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)

Initialize Vgrid object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int `Vgrid_value` (`Vgrid *thee`, double x[3], double *value)

Get potential value (from mesh or approximation) at a point.
- VEXTERNC void `Vgrid_dtor` (`Vgrid **thee`)

Object destructor.
- VEXTERNC void `Vgrid_dtor2` (`Vgrid *thee`)

FORTRAN stub object destructor.
- VEXTERNC int `Vgrid_curvature` (`Vgrid *thee`, double pt[3], int cflag, double *curv)

Get second derivative values at a point.
- VEXTERNC int `Vgrid_gradient` (`Vgrid *thee`, double pt[3], double grad[3])

Get first derivative values at a point.
- VEXTERNC int `Vgrid_readGZ` (`Vgrid *thee`, const char *fname)

Read in OpenDX data in GZIP format.
- VEXTERNC void `Vgrid_writeGZ` (`Vgrid *thee`, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)

Write out OpenDX data in GZIP format.
- VEXTERNC void `Vgrid_writeUHBD` (`Vgrid *thee`, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)

Write out the data in UHBD grid format.
- VEXTERNC void `Vgrid_writeDX` (`Vgrid *thee`, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)

Write out the data in OpenDX grid format.
- VEXTERNC int `Vgrid_readDX` (`Vgrid *thee`, const char *iodev, const char *iofmt, const char *thost, const char *fname)

Read in data in OpenDX grid format.
- VEXTERNC double `Vgrid_integrate` (`Vgrid *thee`)

Get the integral of the data.
- VEXTERNC double `Vgrid_normL1` (`Vgrid *thee`)

Get the L_1 norm of the data. This returns the integral:

$$\|u\|_{L_1} = \int_{\Omega} |u(x)| dx$$

- VEXTERNC double [Vgrid_normL2](#) ([Vgrid *thee](#))

Get the L_2 norm of the data. This returns the integral:

$$\|u\|_{L_2} = \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

- VEXTERNC double [Vgrid_normLinf](#) ([Vgrid *thee](#))

Get the L_∞ norm of the data. This returns the integral:

$$\|u\|_{L_\infty} = \sup_{x \in \Omega} |u(x)|$$

- VEXTERNC double [Vgrid_seminormH1](#) ([Vgrid *thee](#))

Get the H_1 semi-norm of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx \right)^{1/2}$$

- VEXTERNC double [Vgrid_normH1](#) ([Vgrid *thee](#))

Get the H_1 norm (or energy norm) of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx + \int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

10.79.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker and Steve Bond

Version

Id:

[vgrid.h](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [vgrid.h](#).

10.79.2 Function Documentation

10.79.2.1 VEXTERNC void Vgrid_writeGZ (Vgrid * *thee*, const char * *iodev*, const char *
iofmt, const char * *thost*, const char * *fname*, char * *title*, double * *pvec*)

Write out OpenDX data in GZIP format.

Author

Dave Gohara

Parameters

<i>thee</i>	Object to hold new grid data
<i>iodev</i>	I/O device
<i>iofmt</i>	I/O format
<i>thost</i>	Remote host name
<i>fname</i>	File name
<i>title</i>	Data title
<i>pvec</i>	Masking vector (0 = not written)

Definition at line 807 of file vgrid.c.

10.80 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vgrid.h

```
00001
00062 #ifndef _VGRID_H_
00063 #define _VGRID_H_
00064
00065 #include "maloc/malloc.h"
00066 #include "apbs/vhal.h"
00067 #include "apbs/vstring.h"
00068
00069
00072 #define VGRID_DIGITS 6
00073
00079 struct sVgrid {
00080
00081     int nx;
00082     int ny;
00083     int nz;
00084     double hx;
00085     double hy;
00086     double hzed;
00087     double xmin;
00088     double ymin;
00089     double zmin;
00090     double xmax;
00091     double ymax;
00092     double zmax;
```

```

00093     double *data;
00094     int readdata;
00095     int ctordata;
00097     Vmem *mem;
00098 };
00099
00104 typedef struct sVgrid Vgrid;
00105
00106 #if !defined(VINLINE_VGRID)
00107
00115     VEXTERNC unsigned long int Vgrid_memChk(Vgrid *thee);
00116
00117 #else /* if defined(VINLINE_VGRID) */
00118
00126 # define Vgrid_memChk(thee) (Vmem_bytes((thee)->vmem))
00127
00128 #endif /* if !defined(VINLINE_VPMG) */
00129
00147 VEXTERNC Vgrid* Vgrid_ctor(int nx, int ny, int nz,
00148                               double hx, double hy, double hzed,
00149                               double xmin, double ymin, double zmin,
00150                               double *data);
00151
00170 VEXTERNC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int nz,
00171                           double hx, double hy, double hzed,
00172                           double xmin, double ymin, double zmin,
00173                           double *data);
00174
00183 VEXTERNC int Vgrid_value(Vgrid *thee, double x[3], double *value);
00184
00190 VEXTERNC void Vgrid_dtor(Vgrid **thee);
00191
00197 VEXTERNC void Vgrid_dtor2(Vgrid *thee);
00198
00212 VEXTERNC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
00213     double *curv);
00214
00223 VEXTERNC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3] );
00224
00229 VEXTERNC int Vgrid_readGZ(
00230     Vgrid *thee,
00231     const char *fname
00232 );
00233
00237 VEXTERNC void Vgrid_writeGZ(
00238     Vgrid *thee,
00239     const char *iodev,
00240     const char *iofmt,
00241     const char *thost,
00242     const char *fname,
00243     char *title,
00244     double *pvec
00245 );
00246
00264 VEXTERNC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev,
00265     const char *iofmt, const char *thost, const char *fname, char *title,
00266     double *pvec);

```

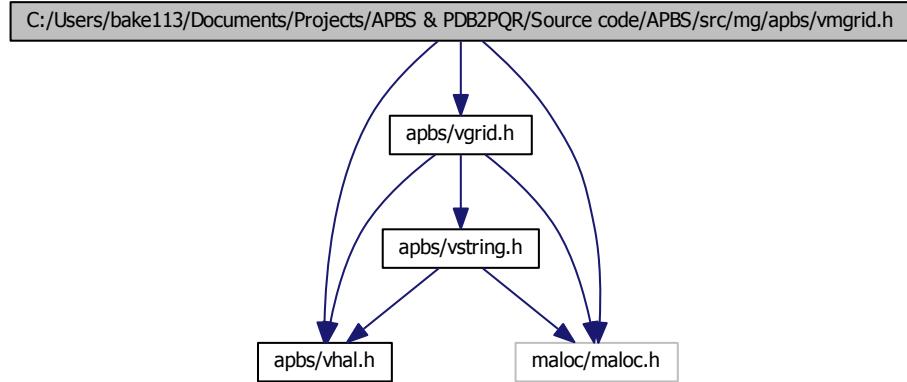
```
00267
00282 VEXTERNC void Vgrid_writeDX(Vgrid *thee, const char *iodev,
00283   const char *icfmt, const char *thost, const char *fname, char *title,
00284   double *pvec);
00285
00297 VEXTERNC int Vgrid_readDX(Vgrid *thee, const char *iodev, const char *iofmt,
00298   const char *thost, const char *fname);
00299
00306 VEXTERNC double Vgrid_integrate(Vgrid *thee);
00307
00316 VEXTERNC double Vgrid_normL1(Vgrid *thee);
00317
00326 VEXTERNC double Vgrid_normL2(Vgrid *thee);
00327
00336 VEXTERNC double Vgrid_normLinf(Vgrid *thee);
00337
00347 VEXTERNC double Vgrid_seminormH1(Vgrid *thee);
00348
00359 VEXTERNC double Vgrid_normH1(Vgrid *thee);
00360
00361 #endif
```

10.81 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vmgrid.h File Reference

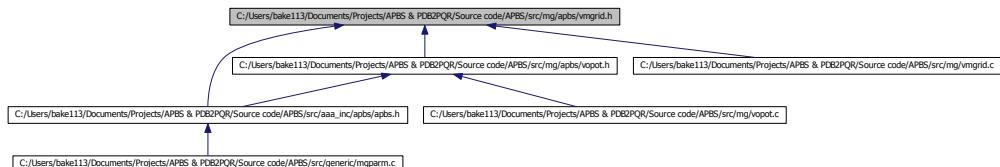
Multiresolution oracle for Cartesian mesh data.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vgrid.h"
```

Include dependency graph for vmgrid.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sVmgrid`

Multiresolution oracle for Cartesian mesh data.

Defines

- #define `VMGRIDMAX` 20

The maximum number of levels in the grid hierarchy.

Typedefs

- **typedef struct sVmgrid Vmgrid**

Declaration of the Vmgrid class as the Vgmrid structure.

Functions

- **VEXTERNC Vmgrid * Vmgrid_ctor ()**
Construct Vmgrid object.
- **VEXTERNC int Vmgrid_ctor2 (Vmgrid *thee)**
Initialize Vmgrid object.
- **VEXTERNC int Vmgrid_value (Vmgrid *thee, double x[3], double *value)**
Get potential value (from mesh or approximation) at a point.
- **VEXTERNC void Vmgrid_dtor (Vmgrid **thee)**
Object destructor.
- **VEXTERNC void Vmgrid_dtor2 (Vmgrid *thee)**
FORTRAN stub object destructor.
- **VEXTERNC int Vmgrid_addGrid (Vmgrid *thee, Vgrid *grid)**
Add a grid to the hierarchy.
- **VEXTERNC int Vmgrid_curvature (Vmgrid *thee, double pt[3], int cflag, double *curv)**
Get second derivative values at a point.
- **VEXTERNC int Vmgrid_gradient (Vmgrid *thee, double pt[3], double grad[3])**
Get first derivative values at a point.
- **VEXTERNC Vgrid * Vmgrid_getGridByNum (Vmgrid *thee, int num)**
Get specific grid in hierarchy.
- **VEXTERNC Vgrid * Vmgrid_getGridByPoint (Vmgrid *thee, double pt[3])**
Get grid in hierarchy which contains specified point or VNULL.

10.81.1 Detailed Description

Multiresolution oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id:

[vmgrid.h](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [vmgrid.h](#).

10.82 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vmgrid.h

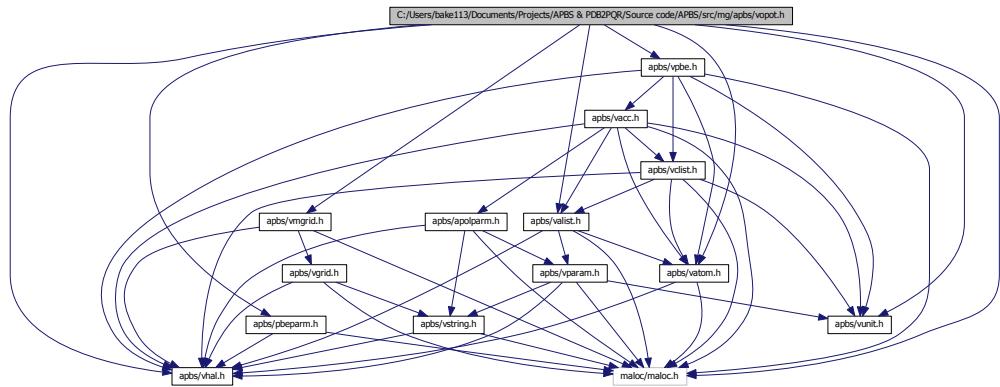
```
00001
00062 #ifndef _VMGRID_H_
00063 #define _VMGRID_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Headers specific to this file */
00070 #include "apbs/vgrid.h"
00071
00076 #define VMGRIDMAX 20
00077
00078
00084 struct sVmgrid {
00085
00086     int ngrids;
00087     Vgrid *grids[VMGRIDMAX];
00092 };
00093
00098 typedef struct sVmgrid Vmgrid;
00099
00105 VEXTERNC Vmgrid* Vmgrid_ctor();
00106
00113 VEXTERNC int Vmgrid_ctor2(Vmgrid *thee);
00114
00123 VEXTERNC int Vmgrid_value(Vmgrid *thee, double x[3], double *value);
00124
00130 VEXTERNC void Vmgrid_dtor(Vmgrid **thee);
00131
00137 VEXTERNC void Vmgrid_dtor2(Vmgrid *thee);
00138
00151 VEXTERNC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid);
00152
00153
00167 VEXTERNC int Vmgrid_curvature(Vmgrid *thee, double pt[3], int cflag,
00168     double *curv);
00169
00178 VEXTERNC int Vmgrid_gradient(Vmgrid *thee, double pt[3], double grad[3] );
00179
00187 VEXTERNC Vgrid* Vmgrid_getGridByNum(Vmgrid *thee, int num);
00188
00196 VEXTERNC Vgrid* Vmgrid_getGridByPoint(Vmgrid *thee, double pt[3]);
00197
00198 #endif
00199
```

10.83 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vopot.h File Reference

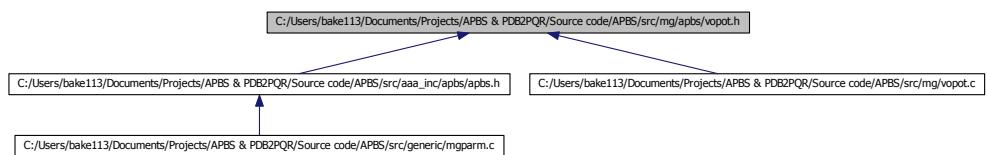
Potential oracle for Cartesian mesh data.

```
#include "maloc/maloc.h"  
#include "apbs/vhal.h"  
#include "apbs/vatom.h"  
#include "apbs/valist.h"  
#include "apbs/vmgrid.h"  
#include "apbs/vunit.h"  
#include "apbs/vpbe.h"  
#include "apbs/pbeparm.h"
```

Include dependency graph for vopot.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **sVopot**
Electrostatic potential oracle for Cartesian mesh data.

Typedefs

- typedef struct **sVopot Vopot**
Declaration of the Vopot class as the Vopot structure.

Functions

- VEXTERNC **Vopot * Vopot_ctor (Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)**
Construct Vopot object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int **Vopot_ctor2 (Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)**
Initialize Vopot object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int **Vopot_pot (Vopot *thee, double x[3], double *pot)**
Get potential value (from mesh or approximation) at a point.
- VEXTERNC void **Vopot_dtor (Vopot **thee)**
Object destructor.
- VEXTERNC void **Vopot_dtor2 (Vopot *thee)**
FORTRAN stub object destructor.
- VEXTERNC int **Vopot_curvature (Vopot *thee, double pt[3], int cflag, double *curv)**
Get second derivative values at a point.
- VEXTERNC int **Vopot_gradient (Vopot *thee, double pt[3], double grad[3])**
Get first derivative values at a point.

10.83.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id:

vopot.h 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [vopot.h](#).

10.84 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vopot.h

```
00001
00062 #ifndef _VOPOT_H_
00063 #define _VOPOT_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Specific headers */
00070 #include "apbs/vatom.h"
00071 #include "apbs/valist.h"
00072 #include "apbs/vmgrid.h"
00073 #include "apbs/vunit.h"
00074 #include "apbs/vpbe.h"
00075 #include "apbs/pbeparm.h"
00076
00082 struct sVopot {
00083
00084     Vmgrid *mgrid;
00086     Vpbe   *pbe;
00087     Vbcfl bcfl;
00089 };
00090
00095 typedef struct sVopot Vopot;
00096
00107 VEXTERNC Vopot* Vopot_ctor(Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl);
00108
00120 VEXTERNC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl);
00121
00130 VEXTERNC int Vopot_pot(Vopot *thee, double x[3], double *pot);
00131
00137 VEXTERNC void Vopot_dtor(Vopot **thee);
00138
00144 VEXTERNC void Vopot_dtor2(Vopot *thee);
00145
00159 VEXTERNC int Vopot_curvature(Vopot *thee, double pt[3], int cflag, double
00160     *curv);
00161
00170 VEXTERNC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3] );
00171
00172
00173 #endif
```

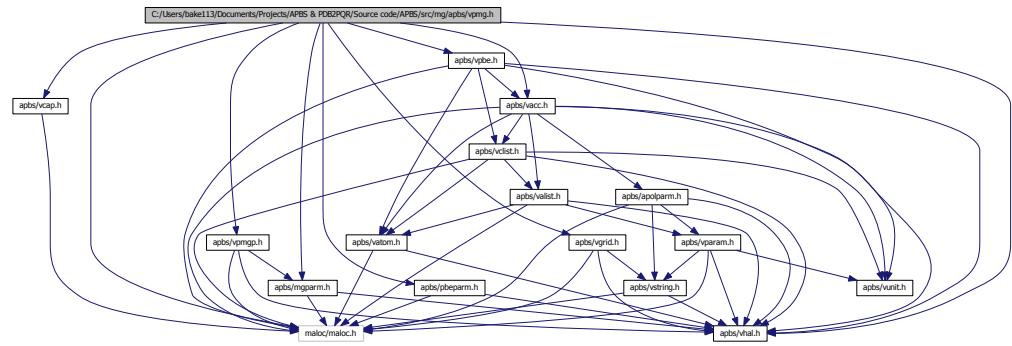
10.85 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vpmg.h File Reference

Contains declarations for class Vpmg.

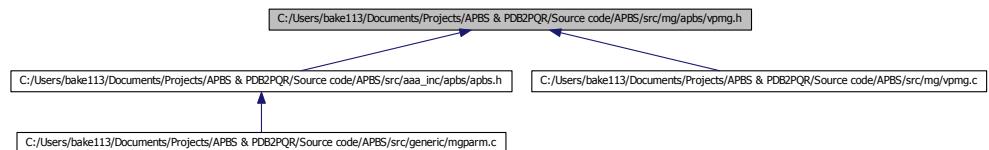
```
#include "maloc/maloc.h"
```

```
#include "apbs/vhal.h"
#include "apbs/vpmgp.h"
#include "apbs/vacc.h"
#include "apbs/vcap.h"
#include "apbs/vpbe.h"
#include "apbs/vgrid.h"
#include "apbs/mgparm.h"
#include "apbs/pbeparm.h"
```

Include dependency graph for vpmg.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct sVpmg

Contains public data members for Vpmg class/module.

Defines

- #define **VPMGMAXPART** 2000

TypeDefs

- typedef struct **sVpmg** **Vpmg**

Declaration of the Vpmg class as the Vpmg structure.

Functions

- VEXTERNC unsigned long int **Vpmg_memChk** (**Vpmg** *thee)
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC **Vpmg** * **Vpmg_ctor** (**Vpmgp** *parms, **Vpbe** *pbe, int focusFlag, **Vpmg** *pmgOLD, **MGparm** *mgparm, **PBEparm_calcEnergy** energyFlag)
Constructor for the Vpmg class (allocates new memory)
- VEXTERNC int **Vpmg_ctor2** (**Vpmg** *thee, **Vpmgp** *parms, **Vpbe** *pbe, int focusFlag, **Vpmg** *pmgOLD, **MGparm** *mgparm, **PBEparm_calcEnergy** energyFlag)
FORTRAN stub constructor for the Vpmg class (uses previously-allocated memory)
- VEXTERNC void **Vpmg_dtor** (**Vpmg** **thee)
Object destructor.
- VEXTERNC void **Vpmg_dtor2** (**Vpmg** *thee)
FORTRAN stub object destructor.
- VEXTERNC int **Vpmg_fillco** (**Vpmg** *thee, **Vsurf_Meth** surfMeth, double splineWin, **Vchrg_Meth** chargeMeth, int useDielXMap, **Vgrid** *dielXMap, int useDielYMap, **Vgrid** *dielYMap, int useDielZMap, **Vgrid** *dielZMap, int useKappaMap, **Vgrid** *kappaMap, int usePotMap, **Vgrid** *potMap, int useChargeMap, **Vgrid** *chargeMap)

Fill the coefficient arrays prior to solving the equation.
- VEXTERNC int **Vpmg_solve** (**Vpmg** *thee)
Solve the PBE using PMG.
- VEXTERNC int **Vpmg_solveLaplace** (**Vpmg** *thee)
Solve Poisson's equation with a homogeneous Laplacian operator using the solvent dielectric constant. This solution is performed by a sine wave decomposition.
- VEXTERNC double **Vpmg_energy** (**Vpmg** *thee, int extFlag)
Get the total electrostatic energy.
- VEXTERNC double **Vpmg_qfEnergy** (**Vpmg** *thee, int extFlag)
Get the "fixed charge" contribution to the electrostatic energy.
- VEXTERNC double **Vpmg_qfAtomEnergy** (**Vpmg** *thee, **Vatom** *atom)
Get the per-atom "fixed charge" contribution to the electrostatic energy.

- VEXTERNC double `Vpmg_qmEnergy` (`Vpmg *thee`, int `extFlag`)

Get the "mobile charge" contribution to the electrostatic energy.
- VEXTERNC double `Vpmg_dielEnergy` (`Vpmg *thee`, int `extFlag`)

Get the "polarization" contribution to the electrostatic energy.
- VEXTERNC double `Vpmg_dielGradNorm` (`Vpmg *thee`)

Get the integral of the gradient of the dielectric function.
- VEXTERNC int `Vpmg_force` (`Vpmg *thee`, double `*force`, int `atomID`, `Vsurf_Meth` `srfm`, `Vchrg_Meth` `chgm`)

Calculate the total force on the specified atom in units of $k_B T/AA$.
- VEXTERNC int `Vpmg_qfForce` (`Vpmg *thee`, double `*force`, int `atomID`, `Vchrg_Meth` `chgm`)

Calculate the "charge-field" force on the specified atom in units of $k_B T/AA$.
- VEXTERNC int `Vpmg_dbForce` (`Vpmg *thee`, double `*dbForce`, int `atomID`, `Vsurf_Meth` `srfm`)

Calculate the dielectric boundary forces on the specified atom in units of $k_B T/AA$.
- VEXTERNC int `Vpmg_ibForce` (`Vpmg *thee`, double `*force`, int `atomID`, `Vsurf_Meth` `srfm`)

Calculate the osmotic pressure on the specified atom in units of $k_B T/AA$.
- VEXTERNC void `Vpmg_setPart` (`Vpmg *thee`, double `lowerCorner[3]`, double `upperCorner[3]`, int `bflags[6]`)

Set partition information which restricts the calculation of observables to a (rectangular) subset of the problem domain.
- VEXTERNC void `Vpmg_unsetPart` (`Vpmg *thee`)

Remove partition restrictions.
- VEXTERNC int `Vpmg_fillArray` (`Vpmg *thee`, double `*vec`, `Vdata_Type` `type`, double `parm`, `Vhal_PBEType` `pbetype`, `PBEmprm` `*pbeparm`)

Fill the specified array with accessibility values.
- VPUBLIC void `Vpmg_fieldSpline4` (`Vpmg *thee`, int `atomID`, double `field[3]`)

Computes the field at an atomic center using a stencil based on the first derivative of a 5th order B-spline.
- VEXTERNC double `Vpmg_qfPermanentMultipoleEnergy` (`Vpmg *thee`, int `atomID`)

Computes the permanent multipole electrostatic hydration energy (the polarization component of the hydration energy currently computed in TINKER).
- VEXTERNC void `Vpmg_qfPermanentMultipoleForce` (`Vpmg *thee`, int `atomID`, double `force[3]`, double `torque[3]`)

Computes the q-Phi Force for permanent multipoles based on 5th order B-splines.
- VEXTERNC void `Vpmg_ibPermanentMultipoleForce` (`Vpmg *thee`, int `atomID`, double `force[3]`)

Compute the ionic boundary force for permanent multipoles.
- VEXTERNC void `Vpmg_dbPermanentMultipoleForce` (`Vpmg *thee`, int `atomID`, double `force[3]`)

Compute the dielectric boundary force for permanent multipoles.

- VEXTERNC void `Vpmg_qfDirectPolForce` (`Vpmg *thee, Vgrid *perm, Vgrid *induced, int atomID, double force[3], double torque[3])`

q-Phi direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void `Vpmg_qfNLDirectPolForce` (`Vpmg *thee, Vgrid *perm, Vgrid *nllInduced, int atomID, double force[3], double torque[3])`

q-Phi direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B-Splines. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void `Vpmg_ibDirectPolForce` (`Vpmg *thee, Vgrid *perm, Vgrid *induced, int atomID, double force[3])`

Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void `Vpmg_ibNLDirectPolForce` (`Vpmg *thee, Vgrid *perm, Vgrid *nllInduced, int atomID, double force[3])`

Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void `Vpmg_dbDirectPolForce` (`Vpmg *thee, Vgrid *perm, Vgrid *induced, int atomID, double force[3])`

Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void `Vpmg_dbNLDirectPolForce` (`Vpmg *thee, Vgrid *perm, Vgrid *nllInduced, int atomID, double force[3])`

Dielectric bounday direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void `Vpmg_qfMutualPolForce` (`Vpmg *thee, Vgrid *induced, Vgrid *nllInduced, int atomID, double force[3])`

Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void `Vpmg_ibMutualPolForce` (`Vpmg *thee, Vgrid *induced, Vgrid *nllInduced, int atomID, double force[3])`

Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void **Vpmg_dbMutualPolForce** (**Vpmg** *thee, **Vgrid** *induced, **Vgrid** *nInduced, int atomID, double force[3])

Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void **Vpmg_printColComp** (**Vpmg** *thee, char path[72], char title[72], char mxtype[3], int flag)

Print out a column-compressed sparse matrix in Harwell-Boeing format.

10.85.1 Detailed Description

Contains declarations for class Vpmg.

Version

Id:

vpmg.h 1667 2011-12-02 23:22:02Z pcells

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*

```

```
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
```

Definition in file [vpmg.h](#).

10.86 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vpmg.h

```
00001
00065 #ifndef _VPMG_H_
00066 #define _VPMG_H_
00067
00068 /* Generic headers */
00069 #include "maloc/maloc.h"
00070 #include "apbs/vhal.h"
00071
00072 /* Headers specific to this file */
00073 #include "apbs/vpmgp.h"
00074 #include "apbs/vacc.h"
00075 #include "apbs/vcap.h"
00076 #include "apbs/vpbe.h"
00077 #include "apbs/vgrid.h"
00078 #include "apbs/mgparm.h"
00079 #include "apbs/pbeparm.h"
00080
00085 #define VPMGMAXPART 2000
00086
00096 struct sVpmg {
00097
00098     Vmem *vmem;
00099     Vpmgp *pmgp;
00100     Vpbe *pbe;
00102     double *epsx;
00103     double *epsy;
```

```

00104     double *epsz;
00105     double *kappa;
00106     double *pot;
00107     double *charge;
00109     int *iparm;
00110     double *rparm;
00111     int *iwork;
00112     double *rwork;
00113     double *a1cf;
00115     double *a2cf;
00117     double *a3cf;
00119     double *ccf;
00120     double *fcf;
00121     double *tcf;
00122     double *u;
00123     double *xf;
00124     double *yf;
00125     double *zf;
00126     double *gxcf;
00127     double *gycf;
00128     double *gzcf;
00129     double *pvec;
00130     double extDiEnergy;
00132     double extQmEnergy;
00134     double extQfEnergy;
00136     double extNpEnergy;
00138     Vsurf_Meth surfMeth;
00139     double splineWin;
00140     Vchrg_Meth chargeMeth;
00141     Vchrg_Src chargeSrc;
00143     int filled;
00145     int useDielXMap;
00147     Vgrid *dielXMap;
00148     int useDielYMap;
00150     Vgrid *dielYMap;
00151     int useDielZMap;
00153     Vgrid *dielZMap;
00154     int useKappaMap;
00156     Vgrid *kappaMap;
00157     int usePotMap;
00159     Vgrid *potMap;
00161     int useChargeMap;
00163     Vgrid *chargeMap;
00164 };
00165
00170     typedef struct sVpmg Vpmg;
00171
00172 /* //////////////////////////////// */
00175 #if !defined(VINLINE_VPMG)
00176
00183     VEXTERNC unsigned long int Vpmg_memChk(
00184         Vpmg *thee
00185     );
00186
00187 #else /* if defined(VINLINE_VPMG) */
00188
00189 #    define Vpmg_memChk(thee) (Vmem_bytes((thee)->vmem))

```

```
00190
00191 #endif /* if !defined(VINLINE_VPMG) */
00192
00193 /* /////////////////////////////////
00194
00195 VEXTERNC Vpmg* Vpmg_ctor(
00196     Vpmgp *parms,
00197     Vpbe *pbe,
00198     int focusFlag,
00199     Vpmg *pmgOLD,
00200     MGparm *mgparm,
00201     PBEparm_calcEnergy energyFlag
00202 );
00203
00204
00205
00206
00207
00208
00209
00210
00211 VEXTERNC int Vpmg_ctor2(
00212     Vpmg *thee,
00213     Vpmgp *parms,
00214     Vpbe *pbe,
00215     int focusFlag,
00216     Vpmg *pmgOLD,
00217     MGparm *mgparm,
00218     PBEparm_calcEnergy energyFlag
00219 );
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233 VEXTERNC void Vpmg_dtor(
00234     Vpmg **thee
00235 );
00236
00237
00238
00239
00240 VEXTERNC void Vpmg_dtor2(
00241     Vpmg *thee
00242 );
00243
00244
00245
00246
00247
00248 VEXTERNC int Vpmg_fillco(
00249     Vpmg *thee,
00250     Vsurf_Meth surfMeth,
00251     double splineWin,
00252     Vchrg_Meth chargeMeth,
00253     int useDielXMap,
00254     Vgrid *dielXMap,
00255     int useDielyMap,
00256     Vgrid *dielyYMap,
00257     int useDielZMap,
00258     Vgrid *dielZMap,
00259     int useKappaMap,
00260     Vgrid *kappaMap,
00261     int usePotMap,
00262     Vgrid *potMap,
00263     int useChargeMap,
00264     Vgrid *chargeMap
00265 );
00266
00267
00268 VEXTERNC int Vpmg_solve(
00269     Vpmg *thee
00270 );
00271
00272
00273 VEXTERNC int Vpmg_solveLaplace(
00274     Vpmg *thee
```

```
00295      );
00296
00306 VEXTERNC double Vpmg_energy(
00307     Vpmg *thee,
00308     int extFlag
00312 );
00313
00331 VEXTERNC double Vpmg_qfEnergy(
00332     Vpmg *thee,
00333     int extFlag
00337 );
00338
00358 VEXTERNC double Vpmg_qfAtomEnergy(
00359     Vpmg *thee,
00360     Vatom *atom
00361 );
00362
00387 VEXTERNC double Vpmg_qmEnergy(
00388     Vpmg *thee,
00389     int extFlag
00393 );
00394
00395
00414 VEXTERNC double Vpmg_dielEnergy(
00415     Vpmg *thee,
00416     int extFlag
00420 );
00421
00422
00439 VEXTERNC double Vpmg_dielGradNorm(
00440     Vpmg *thee
00441 );
00442
00454 VEXTERNC int Vpmg_force(
00455     Vpmg *thee,
00456     double *force,
00458     int atomID,
00459     Vsurf_Meth srfm,
00460     Vchrg_Meth chgm
00461 );
00462
00474 VEXTERNC int Vpmg_qfForce(
00475     Vpmg *thee,
00476     double *force,
00478     int atomID,
00479     Vchrg_Meth chgm
00480 );
00481
00493 VEXTERNC int Vpmg_dbForce(
00494     Vpmg *thee,
00495     double *dbForce,
00497     int atomID,
00498     Vsurf_Meth srfm
00499 );
00500
00512 VEXTERNC int Vpmg_ibForce(
00513     Vpmg *thee,
```

```
00514     double *force,
00516     int atomID,
00517     Vsurf_Meth srfm
00518 );
00519
00520 VEXTERNC void Vpmg_setPart(
00521     Vpmg *thee,
00522     double lowerCorner[3],
00523     double upperCorner[3],
00524     int bflags[6]
00525 );
00526
00527 VEXTERNC void Vpmg_unsetPart(
00528     Vpmg *thee
00529 );
00530
00531 VEXTERNC int Vpmg_fillArray(
00532     Vpmg *thee,
00533     double *vec,
00534     Vdata_Type type,
00535     double parm,
00536     Vhal_PBEType pbetype,
00537     PBEParm * pbeparm
00538 );
00539
00540 VPUBLIC void Vpmg_fieldSpline4(
00541     Vpmg *thee,
00542     int atomID,
00543     double field[3]
00544 );
00545
00546 VEXTERNC double Vpmg_qfPermanentMultipoleEnergy(
00547     Vpmg *thee,
00548     int atomID
00549 );
00550
00551 VEXTERNC void Vpmg_qfPermanentMultipoleForce(
00552     Vpmg *thee,
00553     int atomID,
00554     double force[3],
00555     double torque[3]
00556 );
00557
00558 VEXTERNC void Vpmg_ibPermanentMultipoleForce(
00559     Vpmg *thee,
00560     int atomID,
00561     double force[3]
00562 );
00563
00564 VEXTERNC void Vpmg_dbPermanentMultipoleForce(
00565     Vpmg *thee,
00566     int atomID,
00567     double force[3]
00568 );
00569
00570 VEXTERNC void Vpmg_qfDirectPolForce(
00571     Vpmg *thee,
```

```
00621             Vgrid *perm,
00622             Vgrid *induced,
00623             int atomID,
00624             double force[3],
00625             double torque[3]
00626         );
00627
00636 VEXTERNC void Vpmg_qfNLDirectPolForce(
00637         Vpmg *thee,
00638         Vgrid *perm,
00639         Vgrid *nlInduced,
00640         int atomID,
00641         double force[3],
00642         double torque[3]
00643     );
00644
00652 VEXTERNC void Vpmg_ibDirectPolForce(
00653         Vpmg *thee,
00654         Vgrid *perm,
00655         Vgrid *induced,
00656         int atomID,
00657         double force[3]
00658     );
00659
00668 VEXTERNC void Vpmg_ibNLDirectPolForce(
00669         Vpmg *thee,
00670         Vgrid *perm,
00671         Vgrid *nlInduced,
00672         int atomID,
00673         double force[3]
00674     );
00675
00683 VEXTERNC void Vpmg_dbDirectPolForce(
00684         Vpmg *thee,
00685         Vgrid *perm,
00686         Vgrid *induced,
00687         int atomID,
00688         double force[3]
00689     );
00690
00699 VEXTERNC void Vpmg_dbNLDirectPolForce(
00700         Vpmg *thee,
00701         Vgrid *perm,
00702         Vgrid *nlInduced,
00703         int atomID,
00704         double force[3]
00705     );
00706
00713 VEXTERNC void Vpmg_qfMutualPolForce(
00714         Vpmg *thee,
00715         Vgrid *induced,
00716         Vgrid *nlInduced,
00717         int atomID,
00718         double force[3]
00719     );
00720
00728 VEXTERNC void Vpmg_ibMutualPolForce(
```

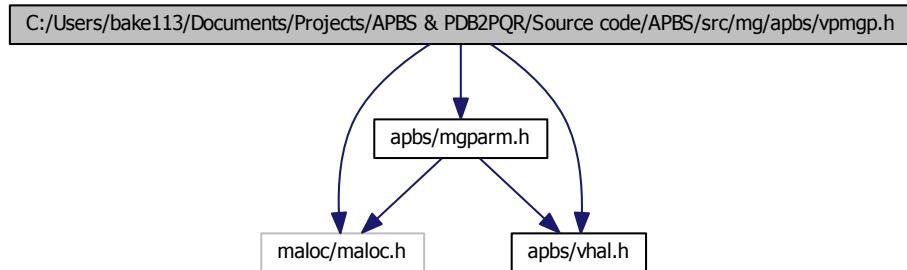
```
00729         Vpmg *thee,
00730         Vgrid *induced,
00731         Vgrid *nlInduced,
00732         int atomID,
00733         double force[3]
00734     );
00735
00743 VEXTERNC void Vpmg_dbMutualPolForce(
00744     Vpmg *thee,
00745     Vgrid *induced,
00746     Vgrid *nlInduced,
00747     int atomID,
00748     double force[3]
00749 );
00750
00757 VEXTERNC void Vpmg_printColComp(
00758     Vpmg *thee,
00759     char path[72],
00760     char title[72],
00761     char mxtype[3],
00762     int flag
00773 );
00774
00775 #endif /* ifndef _VPMG_H_ */
00776
```

10.87 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vpmgp.h File Reference

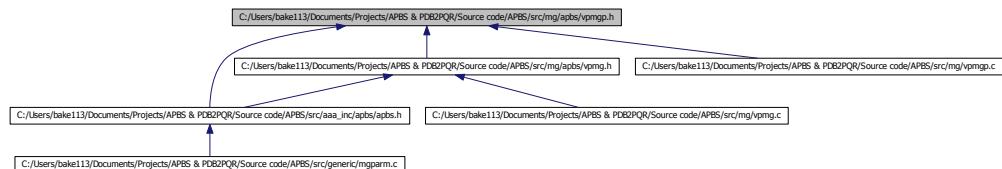
Contains declarations for class Vpmgp.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/mgparm.h"
```

Include dependency graph for vpmgp.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sVpmgp`

Contains public data members for Vpmgp class/module.

Typedefs

- typedef struct `sVpmgp` `Vpmgp`

Declaration of the Vpmgp class as the `sVpmgp` structure.

Functions

- VEXTERNC `Vpmgp *` `Vpmgp_ctor (MGparm *mgparm)`

Construct PMG parameter object and initialize to default values.

- VEXTERNC int [Vpmgp_ctor2](#) ([Vpmgp](#) *thee, [MGparm](#) *mgparm)

FORTRAN stub to construct PMG parameter object and initialize to default values.

- VEXTERNC void [Vpmgp_dtor](#) ([Vpmgp](#) **thee)

Object destructor.

- VEXTERNC void [Vpmgp_dtor2](#) ([Vpmgp](#) *thee)

FORTRAN stub for object destructor.

- VEXTERNC void [Vpmgp_size](#) ([Vpmgp](#) *thee)

Determine array sizes and parameters for multigrid solver.

- VEXTERNC void [Vpmgp_makeCoarse](#) (int numLevel, int nxOld, int nyOld, int nzOld, int *nxNew, int *nyNew, int *nzNew)

Coarsen the grid by the desired number of levels and determine the resulting numbers of grid points.

10.87.1 Detailed Description

Contains declarations for class [Vpmgp](#).

Version

Id:

[vpmgp.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Note

Variables and many default values taken directly from PMG

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*
```

```

* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vpmgp.h](#).

10.88 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/apbs/vpmgp.h

```

00001
00065 #ifndef _VPMGP_H_
00066 #define _VPMGP_H_
00067
00068 #include "maloc/maloc.h"
00069 #include "apbs/vhal.h"
00070 #include "apbs/mgparm.h"
00071
00078 struct sVpmgp {
00079
00080     /* ***** USER-SPECIFIED PARAMETERS ***** */
00081     int nx;
00082     int ny;

```

```
00083     int nz;
00084     int nlev;
00085     double hx;
00086     double hy;
00087     double hzed;
00088     int nonlin;
00089     /* ***** DERIVED PARAMETERS **** */
00090     int nxc;
00091     int nyc;
00092     int nzc;
00093     int nf;
00094     int nc;
00095     int narrc;
00096     int n_rpc;
00097     int n_iz;
00098     int n_ipc;
00099     int nrwk;
00100     int niwk;
00101     int narr;
00102     int ipkey;
00103     /* ***** PARAMETERS WITH DEFAULT VALUES **** */
00104     double xcent;
00105     double ycent;
00106     double zcent;
00107     double errtol;
00108     int itmax;
00109     int istop;
00110     int iinfo;
00111     Vbcfl bcfl;
00112     int key;
00113     int iperf;
00114     int meth;
00115     int mgkey;
00116     int nul;
00117     int nu2;
00118     int mgsmoo;
00119     int mgprol;
00120     int mgcoar;
00121     int mgsolv;
00122     int mgdisc;
00123     double omegal;
00124     double omegan;
00125     int irite;
00126     int ipcon;
00127     double xlen;
00128     double ylen;
00129     double zlen;
00130     double xmin;
00131     double ymin;
00132     double zmin;
00133     double xmax;
00134     double ymax;
00135     double zmax;
00136 };
00137
00202 typedef struct sVpmgp Vpmgp;
00203
```

```

00204 /* ///////////////////////////////// */
00205 // Class Vpmgp: Inlineable methods (vpmgp.c)
00207
00208 #if !defined(VINLINE_VPMGP)
00209 #else /* if defined(VINLINE_VPMGP) */
00210 #endif /* if !defined(VINLINE_VPMGP) */
00211
00212 /* ///////////////////////////////// */
00213 // Class Vpmgp: Non-Inlineable methods (vpmgp.c)
00215
00222 VEXTERNC Vpmgp* Vpmgp_ctor(MGparm *mgparm);
00223
00232 VEXTERNC int Vpmgp_ctor2(Vpmgp *thee, MGparm *mgparm);
00233
00239 VEXTERNC void Vpmgp_dtor(Vpmgp **thee);
00240
00246 VEXTERNC void Vpmgp_dtor2(Vpmgp *thee);
00247
00252 VEXTERNC void Vpmgp_size(
00253     Vpmgp *thee
00254 );
00255
00260 VEXTERNC void Vpmgp_makeCoarse(
00261     int numLevel,
00262     int nxOld,
00263     int nyOld,
00264     int nzOld,
00265     int *nxNew,
00266     int *nyNew,
00267     int *nzNew
00268 );
00269
00270
00271
00272 #endif /* ifndef _VPMGP_H_ */

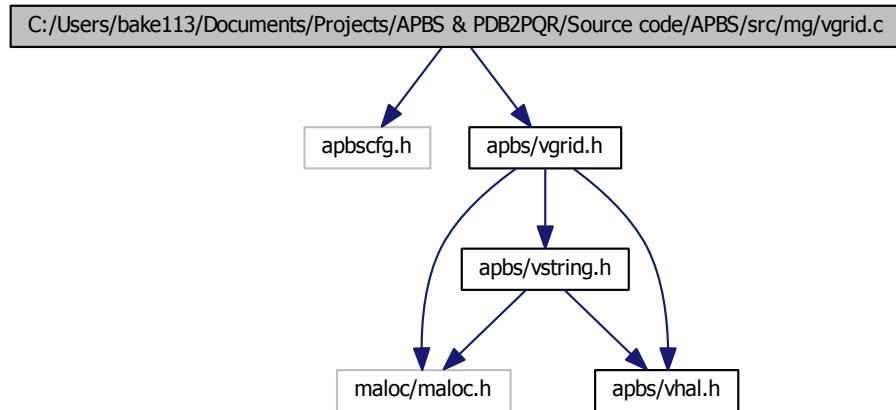
```

10.89 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vgrid.c File Reference

Class Vgrid methods.

```
#include "apbscfg.h"
#include "apbs/vgrid.h"
```

Include dependency graph for vgrid.c:



Defines

- #define **IJK**(i, j, k) (((k)*(nx)*(ny)) + ((j)*(nx)) + (i))

Functions

- VPUBLIC unsigned long int **Vgrid_memChk** (**Vgrid** *thee)

Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC **Vgrid** * **Vgrid_ctor** (int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)

Construct Vgrid object with values obtained from Vpmg_readDX (for example)
- VPUBLIC int **Vgrid_ctor2** (**Vgrid** *thee, int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)

Initialize Vgrid object with values obtained from Vpmg_readDX (for example)
- VPUBLIC void **Vgrid_dtor** (**Vgrid** **thee)

Object destructor.
- VPUBLIC void **Vgrid_dtor2** (**Vgrid** *thee)

FORTTRAN stub object destructor.
- VPUBLIC int **Vgrid_value** (**Vgrid** *thee, double pt[3], double *value)

Get potential value (from mesh or approximation) at a point.

- VPUBLIC int [Vgrid_curvature](#) (**Vgrid** *thee, double pt[3], int cflag, double *value)

Get second derivative values at a point.

- VPUBLIC int [Vgrid_gradient](#) (**Vgrid** *thee, double pt[3], double grad[3])

Get first derivative values at a point.

- VPUBLIC int [Vgrid_readGZ](#) (**Vgrid** *thee, const char *fname)

Read in OpenDX data in GZIP format.

- VPUBLIC int [Vgrid_readDX](#) (**Vgrid** *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

Read in data in OpenDX grid format.

- VPUBLIC void [Vgrid_writeGZ](#) (**Vgrid** *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)

Write out OpenDX data in GZIP format.

- VPUBLIC void [Vgrid_writeDX](#) (**Vgrid** *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)

Write out the data in OpenDX grid format.

- VPUBLIC void [Vgrid_writeUHBD](#) (**Vgrid** *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)

Write out the data in UHBD grid format.

- VPUBLIC double [Vgrid_integrate](#) (**Vgrid** *thee)

Get the integral of the data.

- VPUBLIC double [Vgrid_normL1](#) (**Vgrid** *thee)

Get the L_1 norm of the data. This returns the integral:

$$\|u\|_{L_1} = \int_{\Omega} |u(x)| dx$$

- VPUBLIC double [Vgrid_normL2](#) (**Vgrid** *thee)

Get the L_2 norm of the data. This returns the integral:

$$\|u\|_{L_2} = \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

- VPUBLIC double [Vgrid_seminormH1](#) (**Vgrid** *thee)

Get the H_1 semi-norm of the data. This returns the integral:

$$|u|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx \right)^{1/2}$$

- VPUBLIC double [Vgrid_normH1](#) (**Vgrid** *thee)

Get the H_1 norm (or energy norm) of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx + \int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

- VPUBLIC double [Vgrid_normLinf](#) ([Vgrid](#) *thee)

Get the L_∞ norm of the data. This returns the integral:

$$\|u\|_{L_\infty} = \sup_{x \in \Omega} |u(x)|$$

Variables

- VPRIVATE char * **MCwhiteChars** = " =,;\\t\\n"
- VPRIVATE char * **MCcommChars** = "#%"
- VPRIVATE double **Vcompare**
- VPRIVATE char **Vprecision** [26]

10.89.1 Detailed Description

Class Vgrid methods.

Author

Nathan Baker

Version

Id:

[vgrid.c](#) 1692 2012-01-11 16:18:13Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.
```

```

*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vgrid.c](#).

10.89.2 Function Documentation

10.89.2.1 VPUBLIC void Vgrid_writeGZ (Vgrid **thee*, const char **iodev*, const char **iofmt*,
const char **thost*, const char **fname*, char **title*, double **pvec*)

Write out OpenDX data in GZIP format.

Author

Dave Gohara

Parameters

<i>thee</i>	Object to hold new grid data
<i>iodev</i>	I/O device
<i>iofmt</i>	I/O format
<i>thost</i>	Remote host name
<i>fname</i>	File name
<i>title</i>	Data title
<i>pvec</i>	Masking vector (0 = not written)

Definition at line 807 of file vgrid.c.

10.90 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vgrid.c

```
00001
00057 #include "apbscfg.h"
00058 #include "apbs/vgrid.h"
00059
00060 VEMBED(rcsid="$Id: vgrid.c 1692 2012-01-11 16:18:13Z pcellis $")
00061
00062 #if !defined(VINLINE_VGRID)
00063     VPUBLIC unsigned long int Vgrid_memChk(Vgrid *thee) {
00064         return Vmem_bytes(thee->mem);
00065     }
00066 #endif
00067 #define IJK(i,j,k) (( (k) * (nx) * (ny) )+(( (j) * (nx) )+(i)))
00068
00069 VPRIVATE char *MCwhiteChars = " -; \t\n";
00070 VPRIVATE char *MCommChars = "#%";
00071 VPRIVATE double Vcompare;
00072 VPRIVATE char Vprecision[26];
00073
00074 /* /////////////////////////////////
00075 // Routine: Vgrid_ctor
00076 // Author: Nathan Baker
00077 VPUBLIC Vgrid* Vgrid_ctor(int nx, int ny, int nz,
00078                             double hx, double hy, double hzed,
00079                             double xmin, double ymin, double zmin,
00080                             double *data) {
00081
00082     Vgrid *thee = VNULL;
00083
00084     thee = Vmem_malloc(VNULL, 1, sizeof(Vgrid));
00085     VASSERT(thee != VNULL);
00086     VASSERT(Vgrid_ctor2(thee, nx, ny, nz, hx, hy, hzed,
00087                          xmin, ymin, zmin, data));
00088
00089     return thee;
00090 }
00091 */
00092
00093 /* /////////////////////////////////
00094 // Routine: Vgrid_ctor2
00095 // Author: Nathan Baker
00096 VPUBLIC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int nz,
00097                           double hx, double hy, double hzed,
00098                           double xmin, double ymin, double zmin,
00099                           double *data) {
00100
00101     if (thee == VNULL) return 0;
00102     thee->nx = nx;
00103     thee->ny = ny;
00104     thee->nz = nz;
00105     thee->hx = hx;
```

```

00107     thee->hy = hy;
00108     thee->hzed = hzed;
00109     thee->xmin = xmin;
00110     thee->xmax = xmin + (nx-1)*hx;
00111     thee->ymin = ymin;
00112     thee->ymax = ymin + (ny-1)*hy;
00113     thee->zmin = zmin;
00114     thee->zmax = zmin + (nz-1)*hzed;
00115     if (data == VNULL) {
00116         thee->ctordata = 0;
00117         thee->readdata = 0;
00118     } else {
00119         thee->ctordata = 1;
00120         thee->readdata = 0;
00121         thee->data = data;
00122     }
00123
00124     thee->mem = Vmem_ctor("APBS:VGRID");
00125
00126     Vcompare = pow(10,-1*(VGRID_DIGITS - 2));
00127     sprintf(Vprecision,"%%12.%de %%12.%de %%12.%de", VGRID_DIGITS,
00128             VGRID_DIGITS, VGRID_DIGITS);
00129
00130     return 1;
00131 }
00132
00133 /* //////////////////////////////// */
00134 // Routine: Vgrid_dtor
00135 // Author: Nathan Baker
00137 VPUBLIC void Vgrid_dtor(Vgrid **thee) {
00138
00139     if ((*thee) != VNULL) {
00140         Vgrid_dtor2(*thee);
00141         Vmem_free(VNULL, 1, sizeof(Vgrid), (void **)thee);
00142         (*thee) = VNULL;
00143     }
00144 }
00145
00146 /* //////////////////////////////// */
00147 // Routine: Vgrid_dtor2
00148 // Author: Nathan Baker
00150 VPUBLIC void Vgrid_dtor2(Vgrid *thee) {
00151
00152     if (thee->readdata) {
00153         Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00154             (void **)&(thee->data));
00155     }
00156     Vmem_dtor(&(thee->mem));
00157
00158 }
00159
00160 /* //////////////////////////////// */
00161 // Routine: Vgrid_value
00162 // Author: Nathan Baker
00164 VPUBLIC int Vgrid_value(Vgrid *thee, double pt[3], double *value) {
00165
00166     int nx, ny, nz, ihi, jhi, khi, ilo, jlo, klo;

```

```

00167     double hx, hy, hzed, xmin, ymin, zmin, ifloat, jfloat, kfloat;
00168     double xmax, ymax, zmax;
00169     double u, dx, dy, dz;
00170
00171     if (thee == VNULL) {
00172         Vnm_print(2, "Vgrid_value: Error -- got VNULL thee!\n");
00173         VASSERT(0);
00174     }
00175     if (!(thee->ctordata || thee->readdata)) {
00176         Vnm_print(2, "Vgrid_value: Error -- no data available!\n");
00177         VASSERT(0);
00178     }
00179
00180     nx = thee->nx;
00181     ny = thee->ny;
00182     nz = thee->nz;
00183     hx = thee->hx;
00184     hy = thee->hy;
00185     hzed = thee->hzed;
00186     xmin = thee->xmin;
00187     ymin = thee->ymin;
00188     zmin = thee->zmin;
00189     xmax = thee->xmax;
00190     ymax = thee->ymax;
00191     zmax = thee->zmax;
00192
00193     u = 0;
00194
00195     ifloat = (pt[0] - xmin)/hx;
00196     jfloat = (pt[1] - ymin)/hy;
00197     kfloat = (pt[2] - zmin)/hzed;
00198
00199     ihi = (int)ceil(ifloat);
00200     jhi = (int)ceil(jfloat);
00201     khi = (int)ceil(kfloat);
00202     ilo = (int)floor(ifloat);
00203     jlo = (int)floor(jfloat);
00204     klo = (int)floor(kfloat);
00205     if (VABS(pt[0] - xmin) < Vcompare) ilo = 0;
00206     if (VABS(pt[1] - ymin) < Vcompare) jlo = 0;
00207     if (VABS(pt[2] - zmin) < Vcompare) klo = 0;
00208     if (VABS(pt[0] - xmax) < Vcompare) ihi = nx-1;
00209     if (VABS(pt[1] - ymax) < Vcompare) jhi = ny-1;
00210     if (VABS(pt[2] - zmax) < Vcompare) khi = nz-1;
00211
00212     /* See if we're on the mesh */
00213     if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
00214         (ilo>=0) && (jlo>=0) && (klo>=0)) {
00215
00216         dx = ifloat - (double)(ilo);
00217         dy = jfloat - (double)(jlo);
00218         dz = kfloat - (double)(klo);
00219         u = dx      *dy      *dz      *(thee->data[IJK(ihi,jhi,khi)])
00220         + dx      *(1.0-dy)*dz      *(thee->data[IJK(ihi,jlo,khi)])
00221         + dx      *dy      *(1.0-dz)*(thee->data[IJK(ihi,jhi,klo)])
00222         + dx      *(1.0-dy)*(1.0-dz)*(thee->data[IJK(ihi,jlo,klo)])
00223         + (1.0-dx)*dy      *dz      *(thee->data[IJK(ilo,jhi,khi)])

```

```

00224     + (1.0-dx)*(1.0-dy)*dz      * (thee->data[IJK(ilo,jlo,khi)])
00225     + (1.0-dx)*dy      *(1.0-dz)*(thee->data[IJK(ilo,jhi,klo)])
00226     + (1.0-dx)*(1.0-dy)*(1.0-dz)*(thee->data[IJK(ilo,jlo,klo)]);
00227
00228     *value = u;
00229
00230     if (isnan(u)) {
00231         Vnm_print(2, "Vgrid_value: Got NaN!\n");
00232         Vnm_print(2, "Vgrid_value: (x, y, z) = (%4.3f, %4.3f, %4.3f)\n",
00233             pt[0], pt[1], pt[2]);
00234         Vnm_print(2, "Vgrid_value: (ihi, jhi, khi) = (%d, %d, %d)\n",
00235             ihi, jhi, khi);
00236         Vnm_print(2, "Vgrid_value: (ilo, jlo, klo) = (%d, %d, %d)\n",
00237             ilo, jlo, klo);
00238         Vnm_print(2, "Vgrid_value: (nx, ny, nz) = (%d, %d, %d)\n",
00239             nx, ny, nz);
00240         Vnm_print(2, "Vgrid_value: (dx, dy, dz) = (%4.3f, %4.3f, %4.3f)\n",
00241             dx, dy, dz);
00242         Vnm_print(2, "Vgrid_value: data[IJK(ihi,jhi,khi)] = %g\n",
00243             thee->data[IJK(ihi,jhi,khi)]);
00244         Vnm_print(2, "Vgrid_value: data[IJK(ihi,jlo,khi)] = %g\n",
00245             thee->data[IJK(ihi,jlo,khi)]);
00246         Vnm_print(2, "Vgrid_value: data[IJK(ihi,jhi,klo)] = %g\n",
00247             thee->data[IJK(ihi,jhi,klo)]);
00248         Vnm_print(2, "Vgrid_value: data[IJK(ihi,jlo,klo)] = %g\n",
00249             thee->data[IJK(ihi,jlo,klo)]);
00250         Vnm_print(2, "Vgrid_value: data[IJK(ilo,jhi,khi)] = %g\n",
00251             thee->data[IJK(ilo,jhi,khi)]);
00252         Vnm_print(2, "Vgrid_value: data[IJK(ilo,jlo,khi)] = %g\n",
00253             thee->data[IJK(ilo,jlo,khi)]);
00254         Vnm_print(2, "Vgrid_value: data[IJK(ilo,jhi,klo)] = %g\n",
00255             thee->data[IJK(ilo,jhi,klo)]);
00256         Vnm_print(2, "Vgrid_value: data[IJK(ilo,jlo,klo)] = %g\n",
00257             thee->data[IJK(ilo,jlo,klo)]);
00258     }
00259     return 1;
00260
00261 } else {
00262     *value = 0;
00263     return 0;
00264
00265 }
00266
00267 return 0;
00268
00269
00270 }
00271
00272 /* /////////////////////////////////
00273 // Routine: Vgrid_curvature
00274 //
00275 // Notes: cflag=0 ==> Reduced Maximal Curvature
00276 //          cflag=1 ==> Mean Curvature (Laplace)
00277 //          cflag=2 ==> Gauss Curvature
00278 //          cflag=3 ==> True Maximal Curvature
00279 //

```

```
00280 // Authors: Stephen Bond and Nathan Baker
00281 VPUBLIC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
00282     double *value) {
00283
00284     double hx, hy, hzed, curv;
00285     double dxx, dyy, dzz;
00286     double uleft, umid, uright, testpt[3];
00287
00288     if (thee == VNULL) {
00289         Vnm_print(2, "Vgrid_curvature: Error -- got VNULL thee!\n");
00290         VASSERT(0);
00291     }
00292     if (!(thee->ctordata || thee->readdata)) {
00293         Vnm_print(2, "Vgrid_curvature: Error -- no data available!\n");
00294         VASSERT(0);
00295     }
00296 }
00297
00298     hx = thee->hx;
00299     hy = thee->hy;
00300     hzed = thee->hzed;
00301
00302     curv = 0.0;
00303
00304     testpt[0] = pt[0];
00305     testpt[1] = pt[1];
00306     testpt[2] = pt[2];
00307
00308     /* Compute 2nd derivative in the x-direction */
00309     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00310     testpt[0] = pt[0] - hx;
00311     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00312     testpt[0] = pt[0] + hx;
00313     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00314     testpt[0] = pt[0];
00315
00316     dxx = (uright - 2*umid + uleft)/(hx*hx);
00317
00318     /* Compute 2nd derivative in the y-direction */
00319     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00320     testpt[1] = pt[1] - hy;
00321     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00322     testpt[1] = pt[1] + hy;
00323     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00324     testpt[1] = pt[1];
00325
00326     dyy = (uright - 2*umid + uleft)/(hy*hy);
00327
00328     /* Compute 2nd derivative in the z-direction */
00329     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00330     testpt[2] = pt[2] - hzed;
00331     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00332     testpt[2] = pt[2] + hzed;
00333     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00334
00335     dzz = (uright - 2*umid + uleft)/(hzed*hzed);
```

```

00338     if ( cflag == 0 ) {
00339         curv = fabs(dx);
00340         curv = ( curv > fabs(dy) ) ? curv : fabs(dy);
00341         curv = ( curv > fabs(dz) ) ? curv : fabs(dz);
00342     } else if ( cflag == 1 ) {
00343         curv = (dx + dy + dz)/3.0;
00344     } else {
00345         Vnm_print(2, "Vgrid_curvature: support for cflag = %d not available!\n",
00346             cflag);
00347         VASSERT( 0 ); /* Feature Not Coded Yet! */
00348     }
00349     *value = curv;
00350     return 1;
00351
00352     VERROR1:
00353     return 0;
00354
00355 }
00356
00357 /* /////////////////////////////////
00358 // Routine: Vgrid_gradient
00359 //
00360 // Authors: Nathan Baker and Stephen Bond
00361 VPUBLIC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3]) {
00362
00363     double hx, hy, hzed;
00364     double uleft, umid, uright, testpt[3];
00365     int haveleft, haverright;
00366
00367     if (thee == VNULL) {
00368         Vnm_print(2, "Vgrid_gradient: Error -- got VNULL thee!\n");
00369         VASSERT(0);
00370     }
00371     if (!(thee->ctordata || thee->readdata)) {
00372         Vnm_print(2, "Vgrid_gradient: Error -- no data available!\n");
00373         VASSERT(0);
00374     }
00375
00376     hx = thee->hx;
00377     hy = thee->hy;
00378     hzed = thee->hzed;
00379
00380     /* Compute derivative in the x-direction */
00381     testpt[0] = pt[0];
00382     testpt[1] = pt[1];
00383     testpt[2] = pt[2];
00384     VJMPERR1( Vgrid_value( thee, testpt, &umid));
00385     testpt[0] = pt[0] - hx;
00386     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00387     else haveleft = 0;
00388     testpt[0] = pt[0] + hx;
00389     if (Vgrid_value( thee, testpt, &uright)) haverright = 1;
00390     else haverright = 0;
00391     if (haverright && haveleft) grad[0] = (uright - uleft)/(2*hx);
00392     else if (haverright) grad[0] = (uright - umid)/hx;
00393     else if (haveleft) grad[0] = (umid - uleft)/hx;
00394

```

```

00395     else VJMPERR1(0);
00396
00397     /* Compute derivative in the y-direction */
00398     testpt[0] = pt[0];
00399     testpt[1] = pt[1];
00400     testpt[2] = pt[2];
00401     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00402     testpt[1] = pt[1] - hy;
00403     if (Vgrid_value(thee, testpt, &uleft)) haveleft = 1;
00404     else haveleft = 0;
00405     testpt[1] = pt[1] + hy;
00406     if (Vgrid_value(thee, testpt, &uright)) haveright = 1;
00407     else haveright = 0;
00408     if (haveright && haveleft) grad[1] = (uright - uleft)/(2*hy);
00409     else if (haveright) grad[1] = (uright - umid)/hy;
00410     else if (haveleft) grad[1] = (umid - uleft)/hy;
00411     else VJMPERR1(0);
00412
00413     /* Compute derivative in the z-direction */
00414     testpt[0] = pt[0];
00415     testpt[1] = pt[1];
00416     testpt[2] = pt[2];
00417     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00418     testpt[2] = pt[2] - hzed;
00419     if (Vgrid_value(thee, testpt, &uleft)) haveleft = 1;
00420     else haveleft = 0;
00421     testpt[2] = pt[2] + hzed;
00422     if (Vgrid_value(thee, testpt, &uright)) haveright = 1;
00423     else haveright = 0;
00424     if (haveright && haveleft) grad[2] = (uright - uleft)/(2*hzed);
00425     else if (haveright) grad[2] = (uright - umid)/hzed;
00426     else if (haveleft) grad[2] = (umid - uleft)/hzed;
00427     else VJMPERR1(0);
00428
00429     return 1;
00430
00431     VERROR1:
00432         return 0;
00433
00434 }
00435
00436 /* //////////////////////////////// */
00437 // Routine: Vgrid_readGZ
00438 //
00439 // Author: David Gohara
00440 #ifdef HAVE_ZLIB
00441 #define off_t long
00442 #include "../../../contrib/zlib/zlib.h"
00443 #endif
00444 VPUBLIC int Vgrid_readGZ(Vgrid *thee,
00445                             const char *fname
00446                             )
00447 {
00448
00449 #ifdef HAVE_ZLIB
00450     int i,
00451         j,
00452         k,

```

```

00453             len, /* Temporary counter variable for loop conditionals */
00454             q,
00455             itmp,
00456             u,
00457             header,
00458             incr;
00459     double *temp,
00460             *length,
00461             dtmp1,
00462             dtmp2,
00463             dtmp3;
00464     gzFile infile;
00465     char line[VMAX_ARGLEN];
00466
00467     header = 0;
00468
00469     /* Check to see if the existing data is null and, if not, clear it out */
00470     if (thee->data != VNULL) {
00471         Vnm_print(1, "%s: destroying existing data!\n", __func__);
00472         Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00473             (void **)&(thee->data));
00474     }
00475
00476     thee->readdata = 1;
00477     thee->ctordata = 0;
00478
00479     infile = gzopen(fname, "rb");
00480     if (infile == Z_NULL) {
00481         Vnm_print(2, "%s: Problem opening compressed file %s\n",
00482             __func__, fname);
00483         return VRC_FAILURE;
00484     }
00485
00486     thee->hx = 0.0;
00487     thee->hy = 0.0;
00488     thee->hzd = 0.0;
00489
00490     //read data here
00491     while (header < 7) {
00492         if(gzgets(infile, line, VMAX_ARGLEN) == Z_NULL) {
00493             return VRC_FAILURE;
00494         }
00495
00496         // Skip comments and newlines
00497         if(strncmp(line, "#", 1) == 0) continue;
00498         if(line[0] == '\n') continue;
00499
00500         switch (header) {
00501             case 0:
00502                 sscanf(line, "object 1 class gridpositions counts %d %d %d",
00503                     &(thee->nx), &(thee->ny), &(thee->nz));
00504                 break;
00505             case 1:
00506                 sscanf(line, "origin %lf %lf %lf",
00507                     &(thee->xmin), &(thee->ymin), &(thee->zmin));
00508                 break;
00509             case 2:

```

```

00510     case 3:
00511     case 4:
00512         sscanf(line, "delta %lf %lf %lf", &dtmp1, &dtmp2, &dtmp3);
00513         thee->hx += dtmp1;
00514         thee->hy += dtmp2;
00515         thee->hzed += dtmp3;
00516         break;
00517     default:
00518         break;
00519     }
00520
00521     header++;
00522 }
00523
00524 /* Allocate space for the data */
00525     Vnm_print(0, "%s: allocating %d x %d x %d doubles for storage\n",
00526     __func__, thee->nx, thee->ny, thee->nz);
00527     len = thee->nx * thee->ny * thee->nz;
00528
00529     thee->data = VNULL;
00530     thee->data = Vmem_malloc(thee->mem, len, sizeof(double));
00531     if (thee->data == VNULL) {
00532         Vnm_print(2, "%s: Unable to allocate space for data!\n", __func__);
00533         return 0;
00534     }
00535
00536 /* Allocate a temporary buffer to store the compressed
00537 * data into (column major order). Add 2 to ensure the buffer is
00538 * big enough to take extra data on the final read loop.
00539 */
00540 temp = (double *)malloc(len * (2 * sizeof(double)));
00541 length = temp;
00542
00543 for(i=0;i<len;i+=3){
00544     memset(&line, 0, sizeof(line));
00545     gzgets(infile, line, VMAX_ARGLEN);
00546     sscanf(line, "%lf %lf %lf", &temp[i], &temp[i+1], &temp[i+2]);
00547 }
00548
00549 /* Now move the data to row major order */
00550 incr = 0;
00551 for (i=0; i<thee->nx; i++) {
00552     for (j=0; j<thee->ny; j++) {
00553         for (k=0; k<thee->nz; k++) {
00554             u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00555             (thee->data)[u] = temp[incr++];
00556         }
00557     }
00558 }
00559
00560 /* calculate grid maxima */
00561 thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00562 thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
00563 thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;
00564
00565 /* Close off the socket */
00566 gzclose(infile);

```

```

00567     free(temp);
00568 #else
00569
00570     Vnm_print(0, "WARNING\n");
00571     Vnm_print(0, "Vgrid_readGZ: gzip read/write support is disabled in this build\n");
00572
00573     Vnm_print(0, "Vgrid_readGZ: configure and compile without the --disable-zlib fl
00574 ag.\n");
00575     Vnm_print(0, "WARNING\n");
00576 #endif
00577     return VRC_SUCCESS;
00578 }
00579
00580 /* /////////////////////////////////
00581 // Routine: Vgrid_readDX
00582 //
00583 // Author: Nathan Baker
00584 VPUBLIC int Vgrid_readDX(Vgrid *thee, const char *iodev, const char *iofmt,
00585 const char *thost, const char *fname) {
00586
00587     int i, j, k, itmp, u;
00588     double dtmp;
00589     char tok[VMAX_BUFSIZE];
00590     Vio *sock;
00591
00592     /* Check to see if the existing data is null and, if not, clear it out */
00593     if (thee->data != VNNULL) {
00594         Vnm_print(1, "Vgrid_readDX: destroying existing data!\n");
00595         Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00596                     (void **) &(thee->data));
00597         thee->readdata = 1;
00598         thee->ctordata = 0;
00599
00600     /* Set up the virtual socket */
00601     sock = Vio_ctor(iodev, iofmt, thost, fname, "r");
00602     if (sock == VNNULL) {
00603         Vnm_print(2, "Vgrid_readDX: Problem opening virtual socket %s\n",
00604                   fname);
00605         return 0;
00606     }
00607     if (Vio_accept(sock, 0) < 0) {
00608         Vnm_print(2, "Vgrid_readDX: Problem accepting virtual socket %s\n",
00609                   fname);
00610         return 0;
00611     }
00612     Vio_setWhiteChars(sock, MCwhiteChars);
00613     Vio_setCommChars(sock, MCcommChars);
00614
00615     /* Read in the DX regular positions */
00616     /* Get "object" */
00617     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00618     VJMPERR1(!strcmp(tok, "object"));
00619     /* Get "1" */
00620     VJMPERR2(l == Vio_scanf(sock, "%d", &itmp));
00621     /* Get "class" */
00622     VJMPERR2(l == Vio_scanf(sock, "%s", tok));

```

```

00623     VJMPERR1(!strcmp(tok, "class"));
00624     /* Get "gridpositions" */
00625     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00626     VJMPERR1(!strcmp(tok, "gridpositions"));
00627     /* Get "counts" */
00628     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00629     VJMPERR1(!strcmp(tok, "counts"));
00630     /* Get nx */
00631     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00632     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nx)));
00633     /* Get ny */
00634     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00635     VJMPERR1(1 == sscanf(tok, "%d", &(thee->ny)));
00636     /* Get nz */
00637     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00638     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nz)));
00639     Vnm_print(0, "Vgrid_readDX: Grid dimensions %d x %d x %d grid\n",
00640     thee->nx, thee->ny, thee->nz);
00641     /* Get "origin" */
00642     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00643     VJMPERR1(!strcmp(tok, "origin"));
00644     /* Get xmin */
00645     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00646     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->xmin)));
00647     /* Get ymin */
00648     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00649     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->ymin)));
00650     /* Get zmin */
00651     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00652     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->zmin)));
00653     Vnm_print(0, "Vgrid_readDX: Grid origin = (%g, %g, %g)\n",
00654     thee->xmin, thee->ymin, thee->zmin);
00655     /* Get "delta" */
00656     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00657     VJMPERR1(!strcmp(tok, "delta"));
00658     /* Get hx */
00659     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00660     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hx)));
00661     /* Get 0.0 */
00662     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00663     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00664     VJMPERR1(dtmp == 0.0);
00665     /* Get 0.0 */
00666     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00667     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00668     VJMPERR1(dtmp == 0.0);
00669     /* Get "delta" */
00670     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00671     VJMPERR1(!strcmp(tok, "delta"));
00672     /* Get 0.0 */
00673     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00674     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00675     VJMPERR1(dtmp == 0.0);
00676     /* Get hy */
00677     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00678     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hy)));
00679     /* Get 0.0 */

```

```

00680     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00681     VJMPERR1(l == sscanf(tok, "%lf", &dtmp));
00682     VJMPERR1(dtmp == 0.0);
00683     /* Get "delta" */
00684     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00685     VJMPERR1(!strcmp(tok, "delta"));
00686     /* Get 0.0 */
00687     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00688     VJMPERR1(l == sscanf(tok, "%lf", &dtmp));
00689     VJMPERR1(dtmp == 0.0);
00690     /* Get 0.0 */
00691     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00692     VJMPERR1(l == sscanf(tok, "%lf", &dtmp));
00693     VJMPERR1(dtmp == 0.0);
00694     /* Get hz */
00695     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00696     VJMPERR1(l == sscanf(tok, "%lf", &(thee->hzed)));
00697     Vnm_print(0, "Vgrid_readDX: Grid spacings = (%g, %g, %g)\n",
00698             thee->hx, thee->hy, thee->hzed);
00699     /* Get "object" */
00700     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00701     VJMPERR1(!strcmp(tok, "object"));
00702     /* Get "2" */
00703     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00704     /* Get "class" */
00705     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00706     VJMPERR1(!strcmp(tok, "class"));
00707     /* Get "gridconnections" */
00708     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00709     VJMPERR1(!strcmp(tok, "gridconnections"));
00710     /* Get "counts" */
00711     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00712     VJMPERR1(!strcmp(tok, "counts"));
00713     /* Get the dimensions again */
00714     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00715     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00716     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00717     /* Get "object" */
00718     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00719     VJMPERR1(!strcmp(tok, "object"));
00720     /* Get # */
00721     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00722     /* Get "class" */
00723     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00724     VJMPERR1(!strcmp(tok, "class"));
00725     /* Get "array" */
00726     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00727     VJMPERR1(!strcmp(tok, "array"));
00728     /* Get "type" */
00729     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00730     VJMPERR1(!strcmp(tok, "type"));
00731     /* Get "double" */
00732     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00733     VJMPERR1(!strcmp(tok, "double"));
00734     /* Get "rank" */
00735     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00736     VJMPERR1(!strcmp(tok, "rank"));

```

```

00737     /* Get # */
00738     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00739     /* Get "items" */
00740     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00741     VJMPERR1(!strcmp(tok, "items"));
00742     /* Get # */
00743     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00744     VJMPERR1(1 == sscanf(tok, "%d", &itmp));
00745     VJMPERR1(((thee->nx)*(thee->ny)*(thee->nz)) == itmp);
00746     /* Get "data" */
00747     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00748     VJMPERR1(!strcmp(tok, "data"));
00749     /* Get "follows" */
00750     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00751     VJMPERR1(!strcmp(tok, "follows"));
00752
00753     /* Allocate space for the data */
00754     Vnm_print(0, "Vgrid_readDX: allocating %d x %d x %d doubles for storage\n",
00755             thee->nx, thee->ny, thee->nz);
00756     thee->data = VNULL;
00757     thee->data = Vmem_malloc(thee->mem, (thee->nx)*(thee->ny)*(thee->nz),
00758             sizeof(double));
00759     if (thee->data == VNULL) {
00760         Vnm_print(2, "Vgrid_readDX: Unable to allocate space for data!\n");
00761         return 0;
00762     }
00763
00764     for (i=0; i<thee->nx; i++) {
00765         for (j=0; j<thee->ny; j++) {
00766             for (k=0; k<thee->nz; k++) {
00767                 u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00768                 VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00769                 VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00770                 (thee->data)[u] = dtmp;
00771             }
00772         }
00773     }
00774
00775     /* calculate grid maxima */
00776     thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00777     thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
00778     thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;
00779
00780     /* Close off the socket */
00781     Vio_acceptFree(sock);
00782     Vio_dtor(&sock);
00783
00784     return 1;
00785
00786 VERROR1:
00787     Vio_dtor(&sock);
00788     Vnm_print(2, "Vgrid_readDX: Format problem with input file <%s>\n",
00789             fname);
00790     return 0;
00791
00792 VERROR2:
00793     Vio_dtor(&sock);

```

```

00794     Vnm_print(2, "Vgrid_readDX: I/O problem with input file <%s>\n",
00795             fname);
00796     return 0;
00797
00798
00799
00800 }
00801
00802 /* /////////////////////////////////
00803 // Routine: Vgrid_writeGZ
00804 //
00805 // Author: Nathan Baker
00806 VPUBLIC void Vgrid_writeGZ(Vgrid *thee, const char *iodev, const char *iofmt,
00807                             const char *thost, const char *fname, char *title, double *pvec) {
00808
00809 #ifdef HAVE_ZLIB
00810     double xmin, ymin, zmin, hx, hy, hzed;
00811
00812     int nx, ny, nz;
00813     int icol, i, j, k, u, usepart, nxPART, nyPART, nzPART, gotit;
00814     double x, y, z, xminPART, yminPART, zminPART;
00815
00816     int txyz;
00817     double txmin, tymin, tzmin;
00818
00819     char header[8196];
00820     char footer[8196];
00821     char line[80];
00822     char newline[] = "\n";
00823     gzFile outfile;
00824     char precFormat[VMAX_BUFSIZE];
00825
00826     if (thee == VNULL) {
00827         Vnm_print(2, "Vgrid_writeGZ: Error -- got VNULL thee!\n");
00828         VASSERT(0);
00829     }
00830     if (!(thee->ctordata || thee->readdata)) {
00831         Vnm_print(2, "Vgrid_writeGZ: Error -- no data available!\n");
00832         VASSERT(0);
00833     }
00834 }
00835
00836     hx = thee->hx;
00837     hy = thee->hy;
00838     hzed = thee->hzed;
00839     nx = thee->nx;
00840     ny = thee->ny;
00841     nz = thee->nz;
00842     xmin = thee->xmin;
00843     ymin = thee->ymin;
00844     zmin = thee->zmin;
00845
00846     if (pvec == VNULL) usepart = 0;
00847     else usepart = 1;
00848
00849 /* Set up the virtual socket */
00850 Vnm_print(0, "Vgrid_writeGZ: Opening file...\n");
00851 outfile = gzopen(fname, "wb");

```

```

00852
00853     if (usepart) {
00854         /* Get the lower corner and number of grid points for the local
00855          * partition */
00856         xminPART = VLARGE;
00857         yminPART = VLARGE;
00858         zminPART = VLARGE;
00859         nxPART = 0;
00860         nyPART = 0;
00861         nzPART = 0;
00862         /* First, search for the lower corner */
00863         for (k=0; k<nz; k++) {
00864             z = k*hzed + zmin;
00865             for (j=0; j<ny; j++) {
00866                 y = j*hy + ymin;
00867                 for (i=0; i<nx; i++) {
00868                     x = i*hx + xmin;
00869                     if (pvec[IJK(i,j,k)] > 0.0) {
00870                         if (x < xminPART) xminPART = x;
00871                         if (y < yminPART) yminPART = y;
00872                         if (z < zminPART) zminPART = z;
00873                     }
00874                 }
00875             }
00876         }
00877         /* Now search for the number of grid points in the z direction */
00878         for (k=0; k<nz; k++) {
00879             gotit = 0;
00880             for (j=0; j<ny; j++) {
00881                 for (i=0; i<nx; i++) {
00882                     if (pvec[IJK(i,j,k)] > 0.0) {
00883                         gotit = 1;
00884                         break;
00885                     }
00886                 }
00887                 if (gotit) break;
00888             }
00889             if (gotit) nzPART++;
00890         }
00891         /* Now search for the number of grid points in the y direction */
00892         for (j=0; j<ny; j++) {
00893             gotit = 0;
00894             for (k=0; k<nz; k++) {
00895                 for (i=0; i<nx; i++) {
00896                     if (pvec[IJK(i,j,k)] > 0.0) {
00897                         gotit = 1;
00898                         break;
00899                     }
00900                 }
00901                 if (gotit) break;
00902             }
00903             if (gotit) nyPART++;
00904         }
00905         /* Now search for the number of grid points in the x direction */
00906         for (i=0; i<nx; i++) {
00907             gotit = 0;
00908             for (k=0; k<nz; k++) {

```

```

00909     for (j=0; j<ny; j++) {
00910         if (pvec[IJK(i,j,k)] > 0.0) {
00911             gotit = 1;
00912             break;
00913         }
00914     }
00915     if (gotit) break;
00916 }
00917     if (gotit) nxPART++;
00918 }
00919
00920     if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
00921     Vnm_print(0, "Vgrid_writeGZ: printing only subset of domain\n");
00922 }
00923
00924 txyz = (nxPART*nyPART*nzPART);
00925 txmin = xminPART;
00926 tymin = yminPART;
00927 tzmin = zminPART;
00928
00929 }else {
00930
00931     txyz = (nx*ny*nz);
00932     txmin = xmin;
00933     tymin = ymin;
00934     tzmin = zmin;
00935 }
00936
00937 /* Write off the title (if we're not XDR) */
00938 sprintf(header,
00939 "# Data from %s\n"
00940 "# \n"
00941 "# %s\n"
00942 "# \n"
00943 "# \n"
00944 "#object 1 class gridpositions counts %i %i %i\n"
00945 "#origin %12.6e %12.6e %12.6e\n"
00946 "#delta %12.6e 0.000000e+00 0.000000e+00\n"
00947 "#delta 0.000000e+00 %12.6e 0.000000e+00\n"
00948 "#delta 0.000000e+00 0.000000e+00 %12.6e\n"
00949 "#object 2 class gridconnections counts %i %i %i\n"
00950 "#object 3 class array type double rank 0 items %i data follows\n",
00951 PACKAGE_STRING,title,nx,ny,nz,txmin,tymin,tzmin,
00952 hx,hy,hzed,nx,ny,nz,txyz);
00953 gzwrite(outfile, header, strlen(header)*sizeof(char));
00954
00955 /* Now write the data */
00956 icol = 0;
00957 for (i=0; i<nx; i++) {
00958     for (j=0; j<ny; j++) {
00959         for (k=0; k<nz; k++) {
00960             u = k*(nx)*(ny)+j*(nx)+i;
00961             if (pvec[u] > 0.0) {
00962                 sprintf(line, "%12.6e ", thee->data[u]);
00963                 gzwrite(outfile, line, strlen(line)*sizeof(char));
00964                 icol++;
00965             if (icol == 3) {

```

```

00966     icol = 0;
00967     gzwrite(outfile, newline, strlen(newline)*sizeof(char));
00968 }
00969 }
00970 }
00971 }
00972 }
00973 if(icol < 3){
00974     char newline[] = "\n";
00975     gzwrite(outfile, newline, strlen(newline)*sizeof(char));
00976 }
00977
00978 /* Create the field */
00979 sprintf(footer, "attribute \"dep\" string \"positions\"\n" \
00980     "object \"regular positions regular connections\" class field\n" \
00981     "component \"positions\" value 1\n" \
00982     "component \"connections\" value 2\n" \
00983     "component \"data\" value 3\n");
00984 gzwrite(outfile, footer, strlen(footer)*sizeof(char));
00985
00986 gzclose(outfile);
00987 #else
00988
00989 Vnm_print(0, "WARNING\n");
00990 Vnm_print(0, "Vgrid_readGZ: gzip read/write support is disabled in this build\n");
00991 Vnm_print(0, "Vgrid_readGZ: configure and compile without the --disable-zlib fl
ag.\n");
00992 Vnm_print(0, "WARNING\n");
00993 #endif
00994 }
00995
00996 /* /////////////////////////////////
00997 // Routine: Vgrid_writeDX
00998 //
00999 // Author: Nathan Baker
01000 VPUBLIC void Vgrid_writeDX(Vgrid *thee, const char *iodev, const char *iofmt,
01001     const char *thost, const char *fname, char *title, double *pvec) {
01002
01003     double xmin, ymin, zmin, hx, hy, hzed;
01004     int nx, ny, nz;
01005     int icol, i, j, k, u, usepart, nxPART, nyPART, nzPART, gotit;
01006     double x, y, z, xminPART, yminPART, zminPART;
01007     Vio *sock;
01008     char precFormat[VMAX_BUFSIZE];
01009
01010     if (thee == VNULL) {
01011         Vnm_print(2, "Vgrid_writeDX: Error -- got VNULL thee!\n");
01012         VASSERT(0);
01013     }
01014     if (!(thee->ctordata || thee->readdata)) {
01015         Vnm_print(2, "Vgrid_writeDX: Error -- no data available!\n");
01016         VASSERT(0);
01017     }
01018
01019     hx = thee->hx;
01020     hy = thee->hy;

```

```

01022     hzed = thee->hzed;
01023     nx = thee->nx;
01024     ny = thee->ny;
01025     nz = thee->nz;
01026     xmin = thee->xmin;
01027     ymin = thee->ymin;
01028     zmin = thee->zmin;
01029
01030     if (pvec == VNULL) usepart = 0;
01031     else usepart = 1;
01032
01033     /* Set up the virtual socket */
01034     Vnm_print(0, "Vgrid_writeDX: Opening virtual socket...\n");
01035     sock = Vio_ctor(iodev, iofmt, thost, fname, "w");
01036     if (sock == VNULL) {
01037         Vnm_print(2, "Vgrid_writeDX: Problem opening virtual socket %s\n",
01038             fname);
01039         return;
01040     }
01041     if (Vio_connect(sock, 0) < 0) {
01042         Vnm_print(2, "Vgrid_writeDX: Problem connecting virtual socket %s\n",
01043             fname);
01044         return;
01045     }
01046
01047     Vio_setWhiteChars(sock, MCwhiteChars);
01048     Vio_setCommChars(sock, MCcommChars);
01049
01050     Vnm_print(0, "Vgrid_writeDX: Writing to virtual socket...\n");
01051
01052     if (usepart) {
01053         /* Get the lower corner and number of grid points for the local
01054          * partition */
01055         xminPART = VLARGE;
01056         yminPART = VLARGE;
01057         zminPART = VLARGE;
01058         nxPART = 0;
01059         nyPART = 0;
01060         nzPART = 0;
01061         /* First, search for the lower corner */
01062         for (k=0; k<nz; k++) {
01063             z = k*hzed + zmin;
01064             for (j=0; j<ny; j++) {
01065                 y = j*hy + ymin;
01066                 for (i=0; i<nx; i++) {
01067                     x = i*hx + xmin;
01068                     if (pvec[IJK(i,j,k)] > 0.0) {
01069                         if (x < xminPART) xminPART = x;
01070                         if (y < yminPART) yminPART = y;
01071                         if (z < zminPART) zminPART = z;
01072                     }
01073                 }
01074             }
01075         }
01076         /* Now search for the number of grid points in the z direction */
01077         for (k=0; k<nz; k++) {
01078             gotit = 0;

```

```

01079         for (j=0; j<ny; j++) {
01080             for (i=0; i<nx; i++) {
01081                 if (pvec[IJK(i,j,k)] > 0.0) {
01082                     gotit = 1;
01083                     break;
01084                 }
01085             }
01086             if (gotit) break;
01087         }
01088         if (gotit) nzPART++;
01089     }
01090     /* Now search for the number of grid points in the y direction */
01091     for (j=0; j<ny; j++) {
01092         gotit = 0;
01093         for (k=0; k<nz; k++) {
01094             for (i=0; i<nx; i++) {
01095                 if (pvec[IJK(i,j,k)] > 0.0) {
01096                     gotit = 1;
01097                     break;
01098                 }
01099             }
01100             if (gotit) break;
01101         }
01102         if (gotit) nyPART++;
01103     }
01104     /* Now search for the number of grid points in the x direction */
01105     for (i=0; i<nx; i++) {
01106         gotit = 0;
01107         for (k=0; k<nz; k++) {
01108             for (j=0; j<ny; j++) {
01109                 if (pvec[IJK(i,j,k)] > 0.0) {
01110                     gotit = 1;
01111                     break;
01112                 }
01113             }
01114             if (gotit) break;
01115         }
01116         if (gotit) nxPART++;
01117     }
01118
01119     if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
01120         Vnm_print(0, "Vgrid_writeDX: printing only subset of domain\n");
01121     }
01122
01123
01124     /* Write off the title (if we're not XDR) */
01125     if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01126         Vnm_print(0, "Vgrid_writeDX: Skipping comments for XDR format.\n");
01127     } else {
01128         Vnm_print(0, "Vgrid_writeDX: Writing comments for %s format.\n",
01129                   iofmt);
01130         Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
01131         Vio_printf(sock, "# \n");
01132         Vio_printf(sock, "# %s\n", title);
01133         Vio_printf(sock, "# \n");
01134     }
01135

```

```

01136     /* Write off the DX regular positions */
01137     Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
01138                 nxPART, nyPART, nzPART);
01139
01140     sprintf(precFormat, Vprecision, xminPART, yminPART, zminPART);
01141     Vio_printf(sock, "origin %s\n", precFormat);
01142     sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01143     Vio_printf(sock, "delta %s\n", precFormat);
01144     sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01145     Vio_printf(sock, "delta %s\n", precFormat);
01146     sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01147     Vio_printf(sock, "delta %s\n", precFormat);
01148
01149     /* Write off the DX regular connections */
01150     Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
01151                 nxPART, nyPART, nzPART);
01152
01153     /* Write off the DX data */
01154     Vio_printf(sock, "object 3 class array type double rank 0 items %d \
01155 data follows\n", (nxPART*nyPART*nzPART));
01156     icol = 0;
01157     for (i=0; i<nx; i++) {
01158         for (j=0; j<ny; j++) {
01159             for (k=0; k<nz; k++) {
01160                 u = k*(nx)*(ny)+j*(nx)+i;
01161                 if (pvec[u] > 0.0) {
01162                     Vio_printf(sock, "%12.6e ", thee->data[u]);
01163                     icol++;
01164                     if (icol == 3) {
01165                         icol = 0;
01166                         Vio_printf(sock, "\n");
01167                     }
01168                 }
01169             }
01170         }
01171     }
01172
01173     if (icol != 0) Vio_printf(sock, "\n");
01174
01175     /* Create the field */
01176     Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
01177     Vio_printf(sock, "object \"regular positions regular connections\" \
01178 class field\n");
01179     Vio_printf(sock, "component \"positions\" value 1\n");
01180     Vio_printf(sock, "component \"connections\" value 2\n");
01181     Vio_printf(sock, "component \"data\" value 3\n");
01182
01183 } else {
01184     /* Write off the title (if we're not XDR) */
01185     if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01186         Vnm_print(0, "Vgrid_writeDX: Skipping comments for XDR format.\n");
01187     } else {
01188         Vnm_print(0, "Vgrid_writeDX: Writing comments for %s format.\n",
01189                   iofmt);
01190         Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
01191         Vio_printf(sock, "# \n");
01192         Vio_printf(sock, "# %s\n", title);

```

```

01193         Vio_printf(sock, "# \n");
01194     }
01195
01196
01197     /* Write off the DX regular positions */
01198     Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
01199                nx, ny, nz);
01200
01201     sprintf(precFormat, Vprecision, xmin, ymin, zmin);
01202     Vio_printf(sock, "origin %s\n", precFormat);
01203     sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01204     Vio_printf(sock, "delta %s\n", precFormat);
01205     sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01206     Vio_printf(sock, "delta %s\n", precFormat);
01207     sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01208     Vio_printf(sock, "delta %s\n", precFormat);
01209
01210
01211     /* Write off the DX regular connections */
01212     Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
01213                nx, ny, nz);
01214
01215     /* Write off the DX data */
01216     Vio_printf(sock, "object 3 class array type double rank 0 items %d \
01217 data follows\n", (nx*ny*nz));
01218     icol = 0;
01219     for (i=0; i<nx; i++) {
01220         for (j=0; j<ny; j++) {
01221             for (k=0; k<nz; k++) {
01222                 u = k*(nx)*(ny)+j*(nx)+i;
01223                 Vio_printf(sock, "%12.6e ", theee->data[u]);
01224                 icol++;
01225                 if (icol == 3) {
01226                     icol = 0;
01227                     Vio_printf(sock, "\n");
01228                 }
01229             }
01230         }
01231     if (icol != 0) Vio_printf(sock, "\n");
01232
01233     /* Create the field */
01234     Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
01235     Vio_printf(sock, "object \"regular positions regular connections\" \
01236 class field\n");
01237     Vio_printf(sock, "component \"positions\" value 1\n");
01238     Vio_printf(sock, "component \"connections\" value 2\n");
01239     Vio_printf(sock, "component \"data\" value 3\n");
01240 }
01241
01242     /* Close off the socket */
01243     Vio_connectFree(sock);
01244     Vio_dtor(&sock);
01245 }
01246
01247 /* //////////////////////////////// */
01248 // Routine: Vgrid_writeUHBD
01249 // Author: Nathan Baker

```

```

01251 VPUBLIC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev, const char *iofmt,
01252     const char *thost, const char *fname, char *title, double *pvec) {
01253
01254     int icol, i, j, k, u, nx, ny, nz, gotit;
01255     double xmin, ymin, zmin, hzed, hy, hx;
01256     Vio *sock;
01257
01258     if (thee == VNULL) {
01259         Vnm_print(2, "Vgrid_writeUHBD: Error -- got VNULL thee!\n");
01260         VASSERT(0);
01261     }
01262     if (!(thee->ctordata || thee->readdata)) {
01263         Vnm_print(2, "Vgrid_writeUHBD: Error -- no data available!\n");
01264         VASSERT(0);
01265     }
01266
01267     if ((thee->hx!=thee->hy) || (thee->hy!=thee->hzed)
01268         || (thee->hx!=thee->hzed)) {
01269         Vnm_print(2, "Vgrid_writeUHBD: can't write UHBD mesh with non-uniform \
01270 spacing\n");
01271         return;
01272     }
01273
01274     /* Set up the virtual socket */
01275     sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
01276     if (sock == VNULL) {
01277         Vnm_print(2, "Vgrid_writeUHBD: Problem opening virtual socket %s\n",
01278                 fname);
01279         return;
01280     }
01281     if (Vio_connect(sock, 0) < 0) {
01282         Vnm_print(2, "Vgrid_writeUHBD: Problem connecting virtual socket %s\n",
01283                 fname);
01284         return;
01285     }
01286
01287     /* Get the lower corner and number of grid points for the local
01288      * partition */
01289     hx = thee->hx;
01290     hy = thee->hy;
01291     hzed = thee->hzed;
01292     nx = thee->nx;
01293     ny = thee->ny;
01294     nz = thee->nz;
01295     xmin = thee->xmin;
01296     ymin = thee->ymin;
01297     zmin = thee->zmin;
01298
01299     /* Let interested folks know that partition information is ignored */
01300     if (pvec != VNULL) {
01301         gotit = 0;
01302         for (i=0; i<(nx*ny*nz); i++) {
01303             if (pvec[i] == 0) {
01304                 gotit = 1;
01305                 break;
01306             }
01307         }

```

```

01308     if (gotit) {
01309         Vnm_print(2, "Vgrid_writeUHBD: IGNORING PARTITION INFORMATION!\n");
01310         Vnm_print(2, "Vgrid_writeUHBD: This means I/O from parallel runs \
01311 will have significant overlap.\n");
01312     }
01313 }
01314
01315 /* Write out the header */
01316 Vio_printf(sock, "%72s\n", title);
01317 Vio_printf(sock, "%12.5e%12.5e%7d%7d%7d%7d\n", 1.0, 0.0, -1, 0,
01318     nz, 1, nz);
01319 Vio_printf(sock, "%7d%7d%7d%12.5e%12.5e%12.5e%12.5e\n", nx, ny, nz,
01320     hx, (xmin-hx), (ymin-hx), (zmin-hx));
01321 Vio_printf(sock, "%12.5e%12.5e%12.5e%12.5e\n", 0.0, 0.0, 0.0, 0.0);
01322 Vio_printf(sock, "%12.5e%12.5e%7d%7d", 0.0, 0.0, 0, 0);
01323
01324 /* Write out the entries */
01325 icol = 0;
01326 for (k=0; k<nz; k++) {
01327     Vio_printf(sock, "\n%7d%7d%7d\n", k+1, thee->nx, thee->ny);
01328     icol = 0;
01329     for (j=0; j<ny; j++) {
01330         for (i=0; i<nx; i++) {
01331             u = k*(nx)*(ny)+j*(nx)+i;
01332             icol++;
01333             Vio_printf(sock, " %12.5e", thee->data[u]);
01334             if (icol == 6) {
01335                 icol = 0;
01336                 Vio_printf(sock, "\n");
01337             }
01338         }
01339     }
01340 }
01341 if (icol != 0) Vio_printf(sock, "\n");
01342
01343 /* Close off the socket */
01344 Vio_connectFree(sock);
01345 Vio_dtor(&sock);
01346 }
01347
01348 VPUBLIC double Vgrid_integrate(Vgrid *thee) {
01349
01350     int i, j, k, nx, ny, nz;
01351     double sum, w;
01352
01353     if (thee == VNULL) {
01354         Vnm_print(2, "Vgrid_integrate: Got VNULL thee!\n");
01355         VASSERT(0);
01356     }
01357
01358     nx = thee->nx;
01359     ny = thee->ny;
01360     nz = thee->nz;
01361
01362     sum = 0.0;
01363
01364     for (k=0; k<nz; k++) {

```

```

01365     w = 1.0;
01366     if ((k==0) || (k==(nz-1))) w = w * 0.5;
01367         for (j=0; j<ny; j++) {
01368             w = 1.0;
01369             if ((j==0) || (j==(ny-1))) w = w * 0.5;
01370                 for (i=0; i<nx; i++) {
01371                     w = 1.0;
01372                     if ((i==0) || (i==(nx-1))) w = w * 0.5;
01373                         sum = sum + w*(thee->data[IJK(i,j,k)]);
01374                     }
01375                 }
01376             }
01377             sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01379
01380         return sum;
01381     }
01382 }
01383
01384
01385 VPUBLIC double Vgrid_normL1(Vgrid *thee) {
01386
01387     int i, j, k, nx, ny, nz;
01388     double sum;
01389
01390     if (thee == VNULL) {
01391         Vnm_print(2, "Vgrid_normL1: Got VNULL thee!\n");
01392         VASSERT(0);
01393     }
01394
01395     nx = thee->nx;
01396     ny = thee->ny;
01397     nz = thee->nz;
01398
01399     sum = 0.0;
01400     for (k=0; k<nz; k++) {
01401         for (j=0; j<ny; j++) {
01402             for (i=0; i<nx; i++) {
01403                 sum = sum + VABS(thee->data[IJK(i,j,k)]);
01404             }
01405         }
01406     }
01407     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01408
01409     return sum;
01410 }
01411
01412 }
01413
01414 VPUBLIC double Vgrid_normL2(Vgrid *thee) {
01415
01416     int i, j, k, nx, ny, nz;
01417     double sum;
01418
01419     if (thee == VNULL) {
01420         Vnm_print(2, "Vgrid_normL2: Got VNULL thee!\n");
01421         VASSERT(0);

```

```

01422     }
01423
01424     nx = thee->nx;
01425     ny = thee->ny;
01426     nz = thee->nz;
01427
01428     sum = 0.0;
01429     for (k=0; k<nz; k++) {
01430         for (j=0; j<ny; j++) {
01431             for (i=0; i<nx; i++) {
01432                 sum = sum + VSQR(thee->data[IJK(i,j,k)]);
01433             }
01434         }
01435     }
01436
01437     sum = sum* (thee->hx) * (thee->hy) * (thee->hzed);
01438
01439     return VSQRT(sum);
01440
01441 }
01442
01443 VPUBLIC double Vgrid_seminormH1(Vgrid *thee) {
01444
01445     int i, j, k, d, nx, ny, nz;
01446     double pt[3], grad[3], sum, hx, hy, hzed, xmin, ymin, zmin;
01447
01448     if (thee == VNULL) {
01449         Vnm_print(2, "Vgrid_seminormH1: Got VNULL thee!\n");
01450         VASSERT(0);
01451     }
01452
01453     nx = thee->nx;
01454     ny = thee->ny;
01455     nz = thee->nz;
01456     hx = thee->hx;
01457     hy = thee->hy;
01458     hzed = thee->hzed;
01459     xmin = thee->xmin;
01460     ymin = thee->ymin;
01461     zmin = thee->zmin;
01462
01463     sum = 0.0;
01464     for (k=0; k<nz; k++) {
01465         pt[2] = k*hzed + zmin;
01466         for (j=0; j<ny; j++) {
01467             pt[1] = j*hy + ymin;
01468             for (i=0; i<nx; i++) {
01469                 pt[0] = i*hx + xmin;
01470                 VASSERT(Vgrid_gradient(thee, pt, grad));
01471                 for (d=0; d<3; d++) sum = sum + VSQR(grad[d]);
01472             }
01473         }
01474     }
01475
01476     sum = sum* (hx) * (hy) * (hzed);
01477
01478     if (VABS(sum) < VSMALL) sum = 0.0;

```

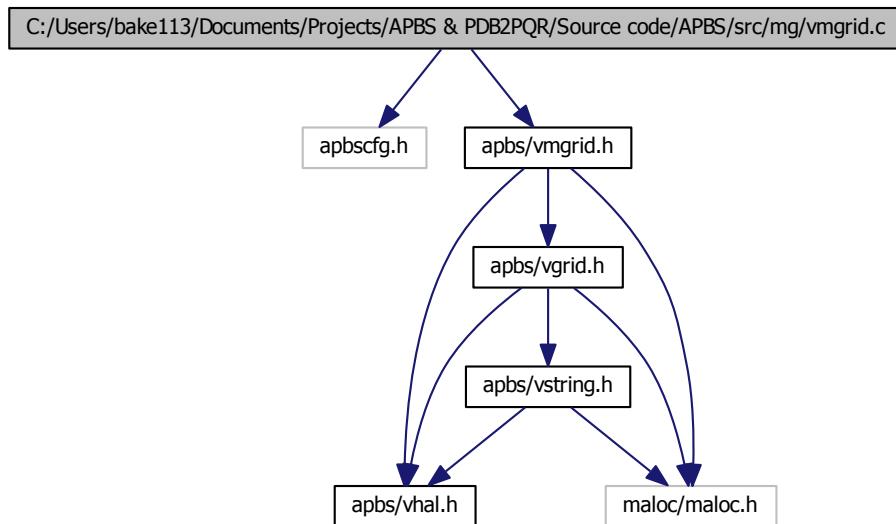
```
01479     else sum = VSQRT(sum);
01480
01481     return sum;
01482
01483 }
01484
01485 VPUBLIC double Vgrid_normH1(Vgrid *thee) {
01486
01487     double sum = 0.0;
01488
01489     if (thee == VNULL) {
01490         Vnm_print(2, "Vgrid_normH1: Got VNULL thee!\n");
01491         VASSERT(0);
01492     }
01493
01494     sum = VSQR(Vgrid_seminormH1(thee)) + VSQR(Vgrid_normL2(thee));
01495
01496     return VSQRT(sum);
01497
01498 }
01499
01500 VPUBLIC double Vgrid_normLinf(Vgrid *thee) {
01501
01502     int i, j, k, nx, ny, nz, gotval;
01503     double sum, val;
01504
01505     if (thee == VNULL) {
01506         Vnm_print(2, "Vgrid_normLinf: Got VNULL thee!\n");
01507         VASSERT(0);
01508     }
01509
01510     nx = thee->nx;
01511     ny = thee->ny;
01512     nz = thee->nz;
01513
01514     sum = 0.0;
01515     gotval = 0;
01516     for (k=0; k<nz; k++) {
01517         for (j=0; j<ny; j++) {
01518             for (i=0; i<nx; i++) {
01519                 val = VABS(thee->data[IJK(i, j, k)]);
01520                 if (!gotval) {
01521                     gotval = 1;
01522                     sum = val;
01523                 }
01524                 if (val > sum) sum = val;
01525             }
01526         }
01527     }
01528
01529     return sum;
01530
01531 }
01532
```

10.91 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vmgrid.c File Reference

Class Vmgrid methods.

```
#include "apbscfg.h"
#include "apbs/vmgrid.h"

Include dependency graph for vmgrid.c:
```



Functions

- VPUBLIC [Vmgrid](#) * [Vmgrid_ctor](#) ()
Construct Vmgrid object.
- VPUBLIC int [Vmgrid_ctor2](#) ([Vmgrid](#) *thee)
Initialize Vmgrid object.
- VPUBLIC void [Vmgrid_dtor](#) ([Vmgrid](#) **thee)
Object destructor.
- VPUBLIC void [Vmgrid_dtor2](#) ([Vmgrid](#) *thee)
FORTTRAN stub object destructor.

- VPUBLIC int [Vmgrid_value](#) ([Vmgrid](#) *thee, double pt[3], double *value)
Get potential value (from mesh or approximation) at a point.
- VPUBLIC int [Vmgrid_curvature](#) ([Vmgrid](#) *thee, double pt[3], int cflag, double *value)
Get second derivative values at a point.
- VPUBLIC int [Vmgrid_gradient](#) ([Vmgrid](#) *thee, double pt[3], double grad[3])
Get first derivative values at a point.
- VPUBLIC int [Vmgrid_addGrid](#) ([Vmgrid](#) *thee, [Vgrid](#) *grid)
Add a grid to the hierarchy.

10.91.1 Detailed Description

Class Vmgrid methods.

Author

Nathan Baker

Version

Id:

[vmgrid.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*

```

```
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
```

Definition in file [vmgrid.c](#).

10.92 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vmgrid.c

```
00001
00057 #include "apbscfg.h"
00058 #include "apbs/vmgrid.h"
00059
00060 VEMBED (rcsid="$Id: vmgrid.c 1667 2011-12-02 23:22:02Z pcellis $")
00061
00062 /* ///////////////////////////////// */
00063 // Routine: Vmgrid_ctor
00064 // Author: Nathan Baker
00065 VPUBLIC Vmgrid* Vmgrid_ctor() {
00066
00067     Vmgrid *thee = VNULL;
00068
00069     thee = Vmem_malloc(VNULL, 1, sizeof(Vmgrid));
00070     VASSERT(thee != VNULL);
00071     VASSERT(Vmgrid_ctor2(thee));
00072
00073     return thee;
00074 }
00075
00076
00077 /* ///////////////////////////////// */
00078 // Routine: Vmgrid_ctor2
00079 // Author: Nathan Baker
```

```

00081 VPUBLIC int Vmgrid_ctor2(Vmgrid *thee) {
00082     int i;
00084
00085     if (thee == VNULL) return 0;
00086
00087     thee->ngrids = 0;
00088     for (i=0; i<VMGRIDMAX; i++) thee->grids[i] = VNULL;
00089
00090     return 1;
00091 }
00092
00093 /* //////////////////////////////// */
00094 // Routine: Vmgrid_dtor
00095 // Author: Nathan Baker
00096 VPUBLIC void Vmgrid_dtor(Vmgrid **thee) {
00097
00098     if ((*thee) != VNULL) {
00099         Vmgrid_dtor2(*thee);
00100         Vmem_free(VNULL, 1, sizeof(Vmgrid), (void **)thee);
00101         (*thee) = VNULL;
00102     }
00103 }
00104
00105
00106 /* //////////////////////////////// */
00107 // Routine: Vmgrid_dtor2
00108 // Author: Nathan Baker
00109 VPUBLIC void Vmgrid_dtor2(Vmgrid *thee) { ; }
00110
00111
00112 /* //////////////////////////////// */
00113 // Routine: Vmgrid_value
00114 // Author: Nathan Baker
00115 VPUBLIC int Vmgrid_value(Vmgrid *thee, double pt[3], double *value) {
00116
00117     int i, rc;
00118     double tvalue;
00119
00120     VASSERT(thee != VNULL);
00121
00122     for (i=0; i<thee->ngrids; i++) {
00123         rc = Vgrid_value(thee->grids[i], pt, &tvalue);
00124         if (rc) {
00125             *value = tvalue;
00126             return 1;
00127         }
00128     }
00129 }
00130
00131     Vnm_print(2, "Vmgrid_value: Point (%g, %g, %g) not found in \
00132 hierarchy!\n", pt[0], pt[1], pt[2]);
00133
00134     return 0;
00135 }
00136
00137 /* //////////////////////////////// */
00138 // Routine: Vmgrid_curvature
00139 //
00140 // Notes: cflag=0 ==> Reduced Maximal Curvature

```

```

00141 //          cflag=1 ==> Mean Curvature (Laplace)
00142 //          cflag=2 ==> Gauss Curvature
00143 //          cflag=3 ==> True Maximal Curvature
00144 //
00145 // Authors: Nathan Baker
00147 VPUBLIC int Vmgrid_curvature(Vmgrid *thee, double pt[3], int cflag,
00148     double *value) {
00149
00150     int i, rc;
00151     double tvalue;
00152
00153     VASSERT(thee != VNULL);
00154
00155     for (i=0; i<thee->ngrids; i++) {
00156         rc = Vgrid_curvature(thee->grids[i], pt, cflag, &tvalue);
00157         if (rc) {
00158             *value = tvalue;
00159             return 1;
00160         }
00161     }
00162
00163     Vnm_print(2, "Vmgrid_curvature: Point (%g, %g, %g) not found in \
00164 hierarchy!\n", pt[0], pt[1], pt[2]);
00165
00166     return 0;
00167
00168
00169 }
00170
00171 /* //////////////////////////////// */
00172 // Routine: Vmgrid_gradient
00173 //
00174 // Authors: Nathan Baker
00176 VPUBLIC int Vmgrid_gradient(Vmgrid *thee, double pt[3], double grad[3]) {
00177
00178     int i, j, rc;
00179     double tgrad[3];
00180
00181     VASSERT(thee != VNULL);
00182
00183     for (i=0; i<thee->ngrids; i++) {
00184         rc = Vgrid_gradient(thee->grids[i], pt, tgrad);
00185         if (rc) {
00186             for (j=0; j<3; j++) grad[j] = tgrad[j];
00187             return 1;
00188         }
00189     }
00190
00191     Vnm_print(2, "Vmgrid_gradient: Point (%g, %g, %g) not found in \
00192 hierarchy!\n", pt[0], pt[1], pt[2]);
00193
00194     return 0;
00195
00196
00197 }
00198
00199 /* //////////////////////////////// */

```

```

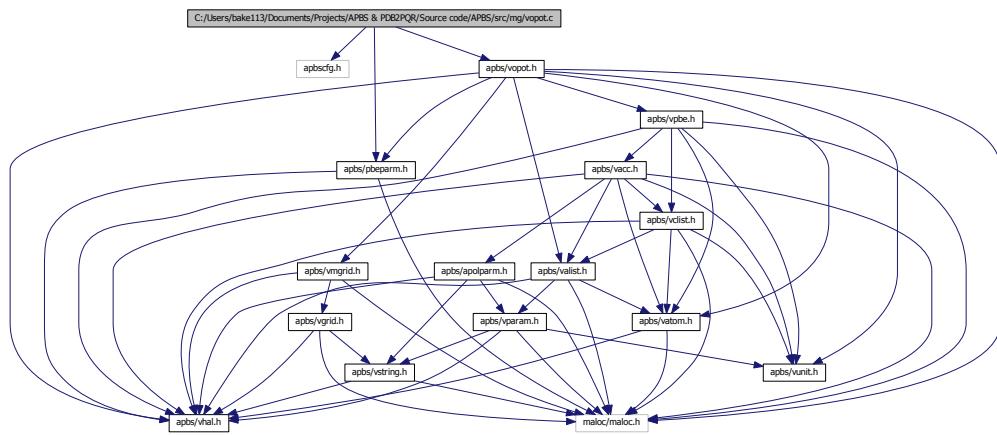
00200 // Routine: Vmgrid_addGrid
00201 //
00202 // Authors: Nathan Baker
00204 VPUBLIC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid) {
00205
00206     int i, j, rc;
00207     double tgrad[3];
00208
00209     VASSERT(thee != VNULL);
00210
00211     if (grid == VNULL) {
00212         Vnm_print(2, "Vmgrid_addGrid: Not adding VNULL grid!\n");
00213         return 0;
00214     }
00215
00216     if (thee->ngrids >= VMGRIDMAX) {
00217         Vnm_print(2, "Vmgrid_addGrid: Too many grids in hierarchy (max = \
00218 %d)\n", VMGRIDMAX);
00219         Vnm_print(2, "Vmgrid_addGrid: Not adding grid!\n");
00220         return 0;
00221     }
00222
00223     thee->grids[thee->ngrids] = grid;
00224     (thee->ngrids)++;
00225
00226     return 1;
00227
00228 }
```

10.93 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vopot.c File Reference

Class Vopot methods.

```
#include "apbscfg.h"
#include "apbs/vopot.h"
#include "apbs/pbeparm.h"
```

Include dependency graph for vopot.c:



Defines

- #define **IJK**(i, j, k) (((k)*(nx)*(ny))+(j)*(nx)+(i))

Functions

- VPUBLIC **Vopot** * **Vopot_ctor** (**Vmgrid** *mgrid, **Vpbe** *pbe, **Vbcfl** bcfl)
Construct Vopot object with values obtained from Vpmg_readDX (for example)
- VPUBLIC int **Vopot_ctor2** (**Vopot** *thee, **Vmgrid** *mgrid, **Vpbe** *pbe, **Vbcfl** bcfl)
Initialize Vopot object with values obtained from Vpmg_readDX (for example)
- VPUBLIC void **Vopot_dtor** (**Vopot** **thee)
Object destructor.
- VPUBLIC void **Vopot_dtor2** (**Vopot** *thee)
FORTRAN stub object destructor.
- VPUBLIC int **Vopot_pot** (**Vopot** *thee, double pt[3], double *value)
Get potential value (from mesh or approximation) at a point.
- VPUBLIC int **Vopot_curvature** (**Vopot** *thee, double pt[3], int cflag, double *value)
Get second derivative values at a point.
- VPUBLIC int **Vopot_gradient** (**Vopot** *thee, double pt[3], double grad[3])
Get first derivative values at a point.

10.93.1 Detailed Description

Class Vopot methods.

Author

Nathan Baker

Version

Id:

[vopot.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vopot.c](#).

10.94 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vopot.c

```
00001
00057 #include "apbscfg.h"
00058 #include "apbs/vopot.h"
00059 #include "apbs/pbeparm.h"
00060
00061 VEMBED (rcsid="$Id: vopot.c 1667 2011-12-02 23:22:02Z pcellis $" )
00062
00063 /* //////////////////////////////// */
00064 // Routine: Vopot_ctor
00065 // Author: Nathan Baker
00067 VPUBLIC Vopot* Vopot_ctor(Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00068
00069     Vopot *thee = VNULL;
00070
00071     thee = Vmem_malloc(VNULL, 1, sizeof(Vopot));
00072     VASSERT(thee != VNULL);
00073     VASSERT(Vopot_ctor2(thee, mgrid, pbe, bcfl));
00074
00075     return thee;
00076 }
00077
00078 /* //////////////////////////////// */
00079 // Routine: Vopot_ctor2
00080 // Author: Nathan Baker
00082 VPUBLIC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00083
00084     if (thee == VNULL) return 0;
00085     thee->bcfl = bcfl;
00086     thee->mgrid = mgrid;
00087     thee->pbe = pbe;
00088
00089     return 1;
00090 }
00091
00092 /* //////////////////////////////// */
00093 // Routine: Vopot_dtor
00094 // Author: Nathan Baker
00096 VPUBLIC void Vopot_dtor(Vopot **thee) {
00097
```

```

00098     if ((*thee) != VNULL) {
00099         Vopot_dtor2(*thee);
00100         Vmem_free(VNULL, 1, sizeof(Vopot), (void **)thee);
00101         (*thee) = VNULL;
00102     }
00103 }
00104
00105 /* /////////////////////////////////
00106 // Routine:  Vopot_dtor2
00107 // Author:   Nathan Baker
00108 VPUBLIC void Vopot_dtor2(Vopot *thee) { return; }
00109
00110 /* /////////////////////////////////
00111 // Routine:  Vopot_pot
00112 // Author:   Nathan Baker
00113 #define IJK(i,j,k) (((k)*(nx)*(ny))+((j)*(nx))+(i))
00114 VPUBLIC int Vopot_pot(Vopot *thee, double pt[3], double *value) {
00115
00116     Vatom *atom;
00117     int i, iatom;
00118     double u, T, charge, eps_w, xkappa, dist, size, val, *position;
00119     Valist *alist;
00120
00121     VASSERT(thee != VNULL);
00122
00123     eps_w = Vpbe_getSolventDiel(thee->pbe);
00124     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00125     T = Vpbe_getTemperature(thee->pbe);
00126     alist = Vpbe_getValist(thee->pbe);
00127
00128     u = 0;
00129
00130     /* See if we're on the mesh */
00131     if (Vmgrid_value(thee->mgrid, pt, &u)) {
00132
00133         *value = u;
00134
00135     } else {
00136
00137         switch (thee->bclf) {
00138
00139             case BCFL_ZERO:
00140                 u = 0;
00141                 break;
00142
00143             case BCFL_SDH:
00144                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00145                 position = Vpbe_getSoluteCenter(thee->pbe);
00146                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00147                 dist = 0;
00148                 for (i=0; i<3; i++)
00149                     dist += VSQR(position[i] - pt[i]);
00150                 dist = (1.0e-10)*VSQRT(dist);
00151                 val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00152                 if (xkappa != 0.0)
00153                     val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00154                 val = val*Vunit_ec/(Vunit_kb*T);
00155
00156

```

```

00157             u = val;
00158             break;
00159
00160         case BCFL_MDH:
00161             u = 0;
00162             for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00163                 atom = Valist_getAtom(alist, iatom);
00164                 position = Vatom_getPosition(atom);
00165                 charge = Vunit_ec*Vatom_getCharge(atom);
00166                 size = (1e-10)*Vatom_getRadius(atom);
00167                 dist = 0;
00168                 for (i=0; i<3; i++)
00169                     dist += VSQR(position[i] - pt[i]);
00170                 dist = (1.0e-10)*VSQRT(dist);
00171                 val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00172                 if (xkappa != 0.0)
00173                     val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00174                 val = val*Vunit_ec/(Vunit_kb*T);
00175                 u = u + val;
00176             }
00177             break;
00178
00179         case BCFL_UNUSED:
00180             Vnm_print(2, "Vopot_pot: Invalid bcfl flag (%d)!\n",
00181                         thee->bcfl);
00182             return 0;
00183
00184         case BCFL_FOCUS:
00185             Vnm_print(2, "Vopot_pot: Invalid bcfl flag (%d)!\n",
00186                         thee->bcfl);
00187             return 0;
00188
00189         default:
00190             Vnm_print(2, "Vopot_pot: Bogus thee->bcfl flag (%d)!\n",
00191                         thee->bcfl);
00192             return 0;
00193             break;
00194     }
00195
00196     *value = u;
00197
00198 }
00199
00200     return 1;
00201
00202 }
00203
00204 /* /////////////////////////////////
00205 // Routine: Vopot_curvature
00206 //
00207 // Notes: cflag=0 ==> Reduced Maximal Curvature
00208 //           cflag=1 ==> Mean Curvature (Laplace)
00209 //           cflag=2 ==> Gauss Curvature
00210 //           cflag=3 ==> True Maximal Curvature
00211 // If we are off the grid, we can still evaluate the Laplacian; assuming, we
00212 // are away from the molecular surface, it is simply equal to the DH factor.
00213 //

```

```

00214 // Authors: Nathan Baker
00216 VPUBLIC int Vopot_curvature(Vopot *thee, double pt[3], int cflag,
00217     double *value) {
00218
00219     Vatom *atom;
00220     int i, iatom;
00221     double u, T, charge, eps_w, xkappa, dist, size, val, *position, zkappa2;
00222     Valist *alist;
00223
00224     VASSERT(thee != VNULL);
00225
00226     eps_w = Vpbe_getSolventDiel(thee->pbe);
00227     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00228     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00229     T = Vpbe_getTemperature(thee->pbe);
00230     alist = Vpbe_getValist(thee->pbe);
00231
00232     u = 0;
00233
00234     if (Vmgrid_curvature(thee->mggrid, pt, cflag, value)) return 1;
00235     else if (cflag != 1) {
00236         Vnm_print(2, "Vopot_curvature: Off mesh!\n");
00237         return 1;
00238     } else {
00239
00240         switch (thee->bcl) {
00241
00242             case BCFL_ZERO:
00243                 u = 0;
00244                 break;
00245
00246             case BCFL_SDH:
00247                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00248                 position = Vpbe_getSoluteCenter(thee->pbe);
00249                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00250                 dist = 0;
00251                 for (i=0; i<3; i++)
00252                     dist += VSQR(position[i] - pt[i]);
00253                 dist = (1.0e-10)*VSQRT(dist);
00254                 if (xkappa != 0.0)
00255                     u = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00256                 break;
00257
00258             case BCFL_MDH:
00259                 u = 0;
00260                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00261                     atom = Valist_getAtom(alist, iatom);
00262                     position = Vatom_getPosition(atom);
00263                     charge = Vunit_ec*Vatom_getCharge(atom);
00264                     size = (1e-10)*Vatom_getRadius(atom);
00265                     dist = 0;
00266                     for (i=0; i<3; i++)
00267                         dist += VSQR(position[i] - pt[i]);
00268                     dist = (1.0e-10)*VSQRT(dist);
00269                     if (xkappa != 0.0)
00270                         val = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00271                     u = u + val;

```

```
00272         }
00273         break;
00274
00275     case BCFL_UNUSED:
00276         Vnm_print(2, "Vopot_pot: Invalid bcfl (%d)!\n", thee->bcfl);
00277         return 0;
00278
00279     case BCFL_FOCUS:
00280         Vnm_print(2, "Vopot_pot: Invalid bcfl (%d)!\n", thee->bcfl);
00281         return 0;
00282
00283     default:
00284         Vnm_print(2, "Vopot_pot: Bogus thee->bcfl flag (%d)!\n",
00285                     thee->bcfl);
00286         return 0;
00287         break;
00288     }
00289
00290     *value = u;
00291 }
00292
00293     return 1;
00294
00295 }
00296
00297 /* /////////////////////////////////
00298 // Routine: Vopot_gradient
00299 //
00300 // Authors: Nathan Baker
00301 VPUBLIC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3]) {
00302
00303     Vatom *atom;
00304     int iatom;
00305     double T, charge, eps_w, xkappa, size, val, *position;
00306     double dx, dy, dz, dist;
00307     Valist *alist;
00308
00309     VASSERT(thee != VNULL);
00310
00311     eps_w = Vpbe_getSolventDiel(thee->pbe);
00312     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00313     T = Vpbe_getTemperature(thee->pbe);
00314     alist = Vpbe_getValist(thee->pbe);
00315
00316
00317     if (!Vmgrid_gradient(thee->mgrid, pt, grad)) {
00318
00319         switch (thee->bcfl) {
00320
00321             case BCFL_ZERO:
00322                 grad[0] = 0.0;
00323                 grad[1] = 0.0;
00324                 grad[2] = 0.0;
00325                 break;
00326
00327             case BCFL_SDH:
00328                 grad[0] = 0.0;
```

```

00330     grad[1] = 0.0;
00331     grad[2] = 0.0;
00332     size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00333     position = Vpbe_getSoluteCenter(thee->pbe);
00334     charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00335     dx = position[0] - pt[0];
00336     dy = position[1] - pt[1];
00337     dz = position[2] - pt[2];
00338     dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00339     dist = (1.0e-10)*VSQRT(dist);
00340     val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00341     if (xkappa != 0.0)
00342         val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00343     val = val*Vunit_ec/(Vunit_kb*T);
00344     grad[0] = val*dx/dist*(-1.0/dist/dist + xkappa/dist);
00345     grad[1] = val*dy/dist*(-1.0/dist/dist + xkappa/dist);
00346     grad[2] = val*dz/dist*(-1.0/dist/dist + xkappa/dist);
00347     break;
00348
00349     case BCFL_MDH:
00350         grad[0] = 0.0;
00351         grad[1] = 0.0;
00352         grad[2] = 0.0;
00353         for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00354             atom = Valist_getAtom(alist, iatom);
00355             position = Vatom_getPosition(atom);
00356             charge = Vunit_ec*Vatom_getCharge(atom);
00357             size = (1e-10)*Vatom_getRadius(atom);
00358             dx = position[0] - pt[0];
00359             dy = position[1] - pt[1];
00360             dz = position[2] - pt[2];
00361             dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00362             dist = (1.0e-10)*VSQRT(dist);
00363             val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00364             if (xkappa != 0.0)
00365                 val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00366             val = val*Vunit_ec/(Vunit_kb*T);
00367             grad[0] += (val*dx/dist*(-1.0/dist/dist + xkappa/dist));
00368             grad[1] += (val*dy/dist*(-1.0/dist/dist + xkappa/dist));
00369             grad[2] += (val*dz/dist*(-1.0/dist/dist + xkappa/dist));
00370         }
00371         break;
00372
00373     case BCFL_UNUSED:
00374         Vnm_print(2, "Vopot: Invalid bcfl (%d)!\n", thee->bcfl);
00375         return 0;
00376
00377     case BCFL_FOCUS:
00378         Vnm_print(2, "Vopot: Invalid bcfl (%d)!\n", thee->bcfl);
00379         return 0;
00380
00381     default:
00382         Vnm_print(2, "Vopot_pot: Bogus thee->bcfl flag (%d)!\n",
00383                   thee->bcfl);
00384         return 0;
00385     break;
00386 }
```

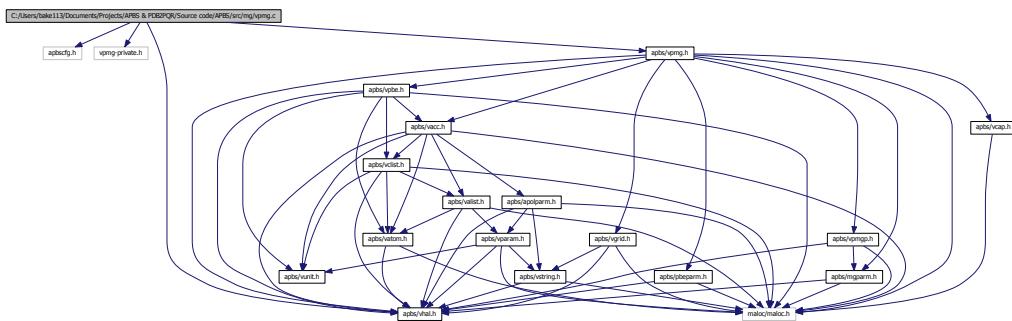
```
00387         return 1;
00388     }
00389 }
00390
00391     return 1;
00392
00393 }
00394
```

10.95 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vpmg.c File Reference

Class Vpmg methods.

```
#include "apbscfg.h"  
#include "vpmg-private.h"  
#include "apbs/vpmg.h"  
#include "apbs/vhal.h"
```

Include dependency graph for vpmg.c:



Functions

- VPUBLIC unsigned long int **Vpmg_memChk** (**Vpmg** *thee)
Return the memory used by this structure (and its contents) in bytes.
 - VPUBLIC void **Vpmg_printColComp** (**Vpmg** *thee, char path[72], char title[72], char mxtype[3], int flag)
Print out a column-compressed sparse matrix in Harwell-Boeing format.
 - VPUBLIC **Vpmg** * **Vpmg_ctor** (**Vpmgp** *pmgp, **Vpb** *pbe, int focusFlag, **Vpmg** *pmgOLD, **MGparm** *mgparm, **PBEparm** calcEnergy energyFlag)

Constructor for the Vpmg class (allocates new memory)

- VPUBLIC int **Vpmg_ctor2** (*Vpmg *thee, Vpmgp *pmgp, Vpbe *pbe, int focusFlag, Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag*)
FORTRAN stub constructor for the Vpmg class (uses previously-allocated memory)
- VPUBLIC int **Vpmg_solve** (*Vpmg *thee*)
Solve the PBE using PMG.
- VPUBLIC void **Vpmg_dtor** (*Vpmg **thee*)
Object destructor.
- VPUBLIC void **Vpmg_dtor2** (*Vpmg *thee*)
FORTRAN stub object destructor.
- VPUBLIC void **Vpmg_setPart** (*Vpmg *thee, double lowerCorner[3], double upperCorner[3], int bflags[6]*)
Set partition information which restricts the calculation of observables to a (rectangular) subset of the problem domain.
- VPUBLIC void **Vpmg_unsetPart** (*Vpmg *thee*)
Remove partition restrictions.
- VPUBLIC int **Vpmg_fillArray** (*Vpmg *thee, double *vec, Vdata_Type type, double parm, Vhal_PBEType pbtype, PBEparm *pbeparm*)
Fill the specified array with accessibility values.
- VPRIVATE double **Vpmg_polarizEnergy** (*Vpmg *thee, int extFlag*)
- VPUBLIC double **Vpmg_energy** (*Vpmg *thee, int extFlag*)
Get the total electrostatic energy.
- VPUBLIC double **Vpmg_dielEnergy** (*Vpmg *thee, int extFlag*)
Get the "polarization" contribution to the electrostatic energy.
- VPUBLIC double **Vpmg_dielGradNorm** (*Vpmg *thee*)
Get the integral of the gradient of the dielectric function.
- VPUBLIC double **Vpmg_qmEnergy** (*Vpmg *thee, int extFlag*)
Get the "mobile charge" contribution to the electrostatic energy.
- VPRIVATE double **Vpmg_qmEnergyNONLIN** (*Vpmg *thee, int extFlag*)
- VPUBLIC double **Vpmg_qmEnergySMPBE** (*Vpmg *thee, int extFlag*)
- VPUBLIC double **Vpmg_qfEnergy** (*Vpmg *thee, int extFlag*)
Get the "fixed charge" contribution to the electrostatic energy.
- VPRIVATE double **Vpmg_qfEnergyPoint** (*Vpmg *thee, int extFlag*)
- VPUBLIC double **Vpmg_qfAtomEnergy** (*Vpmg *thee, Vatom *atom*)
Get the per-atom "fixed charge" contribution to the electrostatic energy.
- VPRIVATE double **Vpmg_qfEnergyVolume** (*Vpmg *thee, int extFlag*)
- VPRIVATE void **Vpmg_splineSelect** (*int srfm, Vacc *acc, double *gpos, double win, double infrad, Vatom *atom, double *force*)
- VPRIVATE void **focusFillBound** (*Vpmg *thee, Vpmg *pmgOLD*)
- VPRIVATE void **extEnergy** (*Vpmg *thee, Vpmg *pmgOLD, PBEparm_calcEnergy extFlag, double partMin[3], double partMax[3], int bflags[6]*)

- VPRIVATE double **bcfl1sp** (double size, double *apos, double charge, double xkappa, double pre1, double *pos)
- VPRIVATE void **bcfl1** (double size, double *apos, double charge, double xkappa, double pre1, double *gxcf, double *gycf, double *gzcf, double *xf, double *yf, double *zf, int nx, int ny, int nz)
- VPRIVATE void **bcfl2** (double size, double *apos, double charge, double *dipole, double *quad, double xkappa, double eps_p, double eps_w, double T, double *gxcf, double *gycf, double *gzcf, double *xf, double *yf, double *zf, int nx, int ny, int nz)
- VPRIVATE void **bcCalcOrig** (**Vpmg** *thee)
- VPRIVATE int **gridPointIsValid** (int i, int j, int k, int nx, int ny, int nz)
- VPRIVATE void **packAtoms** (double *ax, double *ay, double *az, double *charge, double *size, **Vpmg** *thee)
- VPRIVATE void **packUnpack** (int nx, int ny, int nz, int ngrid, double *gx, double *gy, double *gz, double *value, **Vpmg** *thee, int pack)
- VPRIVATE void **bcflnew** (**Vpmg** *thee)
- VPRIVATE void **multipolebc** (double r, double kappa, double eps_p, double eps_w, double rad, double tsr[3])
- VPRIVATE void **bcfl_sdh** (**Vpmg** *thee)
- VPRIVATE void **bcfl_mdh** (**Vpmg** *thee)
- VPRIVATE void **bcfl_mem** (double zmem, double L, double eps_m, double eps_w, double V, double xkappa, double *gxcf, double *gycf, double *gzcf, double *xf, double *yf, double *zf, int nx, int ny, int nz)
- VPRIVATE void **bcfl_map** (**Vpmg** *thee)
- VPRIVATE void **bcCalc** (**Vpmg** *thee)
- VPRIVATE void **fillcoCoefMap** (**Vpmg** *thee)
- VPRIVATE void **fillcoCoefMol** (**Vpmg** *thee)
- VPRIVATE void **fillcoCoefMolOn** (**Vpmg** *thee)
- VPRIVATE void **fillcoCoefMolDiel** (**Vpmg** *thee)
- VPRIVATE void **fillcoCoefMolDielNoSmooth** (**Vpmg** *thee)
- VPRIVATE void **fillcoCoefMolDielSmooth** (**Vpmg** *thee)
- VPRIVATE void **fillcoCoefSpline** (**Vpmg** *thee)
- VPRIVATE void **fillcoCoef** (**Vpmg** *thee)
- VPRIVATE Vrc_Codes **fillcoCharge** (**Vpmg** *thee)
- VPRIVATE Vrc_Codes **fillcoChargeMap** (**Vpmg** *thee)
- VPRIVATE void **fillcoChargeSpline1** (**Vpmg** *thee)
- VPRIVATE double **bspline2** (double x)
- VPRIVATE double **dbspline2** (double x)
- VPRIVATE void **fillcoChargeSpline2** (**Vpmg** *thee)
- VPUBLIC int **Vpmg_fillco** (**Vpmg** *thee, **Vsurf_Meth** surfMeth, double splineWin, **Vchrg_Meth** chargeMeth, int useDielXMap, **Vgrid** *dielXMap, int useDielYMap, **Vgrid** *dielYMap, int useDielZMap, **Vgrid** *dielZMap, int useKappaMap, **Vgrid** *kappaMap, int usePotMap, **Vgrid** *potMap, int useChargeMap, **Vgrid** *chargeMap)

Fill the coefficient arrays prior to solving the equation.

- VPUBLIC int `Vpmg_force` (`Vpmg` *thee, double *force, int atomID, `Vsurf_Meth` srfm, `Vchrg_Meth` chgm)

Calculate the total force on the specified atom in units of k_B T/AA.

- VPUBLIC int `Vpmg_ibForce` (`Vpmg` *thee, double *force, int atomID, `Vsurf_Meth` srfm)

Calculate the osmotic pressure on the specified atom in units of k_B T/AA.

- VPUBLIC int `Vpmg_dbForce` (`Vpmg` *thee, double *dbForce, int atomID, `Vsurf_Meth` srfm)

Calculate the dielectric boundary forces on the specified atom in units of k_B T/AA.

- VPUBLIC int `Vpmg_qfForce` (`Vpmg` *thee, double *force, int atomID, `Vchrg_Meth` chgm)

Calculate the "charge-field" force on the specified atom in units of k_B T/AA.

- VPRIVATE void `qfForceSpline1` (`Vpmg` *thee, double *force, int atomID)
- VPRIVATE void `qfForceSpline2` (`Vpmg` *thee, double *force, int atomID)
- VPRIVATE void `qfForceSpline4` (`Vpmg` *thee, double *force, int atomID)
- VPRIVATE void `markFrac` (double rtot, double *tpos, int nx, int ny, int nz, double hx, double hy, double hz, double xmin, double ymin, double zmin, double *xarray, double *yarray, double *zarray)
- VPRIVATE void `markSphere` (double rtot, double *tpos, int nx, int ny, int nz, double hx, double hy, double hz, double xmin, double ymin, double zmin, double *array, double markVal)
- VPRIVATE void `zlapSolve` (`Vpmg` *thee, double **solution, double **source, double **work1)
- VPUBLIC int `Vpmg_solveLaplace` (`Vpmg` *thee)

Solve Poisson's equation with a homogeneous Laplacian operator using the solvent dielectric constant. This solution is performed by a sine wave decomposition.

- VPRIVATE double `VFCHI4` (int i, double f)
- VPRIVATE double `bspline4` (double x)
- VPUBLIC double `dbspline4` (double x)
- VPUBLIC double `d2bspline4` (double x)
- VPUBLIC double `d3bspline4` (double x)
- VPUBLIC void `fillcoPermanentMultipole` (`Vpmg` *thee)
- VPRIVATE void `fillcoCoefSpline4` (`Vpmg` *thee)
- VPUBLIC void `fillcoPermanentInduced` (`Vpmg` *thee)
- VPRIVATE void `fillcoCoefSpline3` (`Vpmg` *thee)

10.95.1 Detailed Description

Class `Vpmg` methods.

Author

Nathan Baker

Version

Id:

[vpmg.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
```

```
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vpmg.c](#).

10.96 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vpmg.c

```
00001
00057 #include "apbscfg.h"
00058 #include "vpmg-private.h"
00059 #include "apbs/vpmg.h"
00060 #include "apbs/vhal.h"
00061
00062 VMBED(rcsid="$Id: vpmg.c 1667 2011-12-02 23:22:02Z pcellis $")
00063
00064 #if !defined(VINLINE_VPMG)
00065
00066 VPUBLIC unsigned long int Vpmg_memChk(Vpmg *thee) {
00067     if (thee == VNULL) return 0;
00068     return Vmem_bytes(thee->vmem);
00069 }
00070
00071 #endif /* if !defined(VINLINE_VPMG) */
00072
00073
00074 VPUBLIC void Vpmg_printColComp(Vpmg *thee, char path[72], char title[72],
00075     char mxtyp[3], int flag) {
00076
00077     int nn, nxm2, nym2, nzm2, ncol, nrow, nonz;
00078     double *nzval;
00079     int *colptr, *rowind;
00080
00081     /* Calculate the total number of unknowns */
00082     nxm2 = thee->pmgp->nx - 2;
00083     nym2 = thee->pmgp->ny - 2;
00084     nzm2 = thee->pmgp->nz - 2;
00085     nn = nxm2*nym2*nzm2;
00086     ncol = nn;
00087     nrow = nn;
00088
00089     /* Calculate the number of non-zero matrix entries:
00090      *      nn      nonzeros on diagonal
00091      *      nn-1    nonzeros on first off-diagonal
00092      *      nn-nx   nonzeros on second off-diagonal
00093      *      nn-nx*ny nonzeros on third off-diagonal
00094      *
00095      *      7*nn-2*nx*ny-2*nx-2 TOTAL non-zeros
00096      */
00097     nonz = 7*nn - 2*nxm2*nym2 - 2*nxm2 - 2;
00098     nzval = Vmem_malloc(thee->vmem, nonz, sizeof(double));
00099     rowind = Vmem_malloc(thee->vmem, nonz, sizeof(int));
```

```

00100     colptr = Vmem_malloc(thee->vmem, (ncol+1), sizeof(int));
00101
00102 #ifndef VAPBSQUIET
00103     Vnm_print(1, "Vpmg_printColComp: Allocated space for %d nonzeros\n",
00104             nzval);
00105 #endif
00106
00107     F77BCOLCOMP(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00108             nzval, rowind, colptr, &flag);
00109
00110 #if 0
00111     for (i=0; i<nn; i++) {
00112         Vnm_print(1, "nnz(%d) = %g\n", i, nzval[i]);
00113     }
00114 #endif
00115
00116 /* I do not understand why I need to pass nzval in this way, but it
00117 * works... */
00118     F77PCOLCOMP(&nrow, &ncol, &nzval, &(nzval[0]), rowind, colptr, path, title,
00119             mxtype);
00120
00121     Vmem_free(thee->vmem, (ncol+1), sizeof(int), (void **)&colptr);
00122     Vmem_free(thee->vmem, nzval, sizeof(int), (void **)&rowind);
00123     Vmem_free(thee->vmem, nzval, sizeof(double), (void **)&nzval);
00124
00125 }
00126
00127 VPUBLIC Vpmg* Vpmg_ctor(Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
00128                           Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag) {
00129
00130     Vpmg *thee = VNULL;
00131
00132     thee = Vmem_malloc(VNULL, 1, sizeof(Vpmg) );
00133     VASSERT(thee != VNULL);
00134     VASSERT( Vpmg_ctor2(thee, pmgp, pbe, focusFlag, pmgOLD, mgparm,
00135                           energyFlag) );
00136     return thee;
00137 }
00138
00139 VPUBLIC int Vpmg_ctor2(Vpmg *thee, Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
00140                           Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag) {
00141
00142     int i, j, nion;
00143     double ionConc[MAXION], ionQ[MAXION], ionRadii[MAXION], zkappa2, zks2;
00144     double ionstr, partMin[3], partMax[3];
00145
00146 /* Get the parameters */
00147     VASSERT(pmgp != VNULL);
00148     VASSERT(pbe != VNULL);
00149     thee->pmgp = pmgp;
00150     thee->pbe = pbe;
00151
00152 /* Set up the memory */
00153     thee->vmem = Vmem_ctor("APBS:VPMG");
00154
00155 /* TEMPORARY USEAQUA */
00156 /* Calculate storage requirements */

```

```

00157 if (mgparm->useAqua == 0) {
00158   Vpmgp_size(thee->pmgp);
00159 } else {
00160   F77MGSZAQUA (
00161     &(thee->pmgp->mgcoar), &(thee->pmgp->mgdisc),
00162     &(thee->pmgp->mgsolv),
00163     &(thee->pmgp->nx), &(thee->pmgp->ny), &(thee->pmgp->nz),
00164     &(thee->pmgp->nlev),
00165     &(thee->pmgp->nxc), &(thee->pmgp->nyc), &(thee->pmgp->nzc),
00166     &(thee->pmgp->nf), &(thee->pmgp->nc),
00167     &(thee->pmgp->narr), &(thee->pmgp->narrc),
00168     &(thee->pmgp->n_rpc), &(thee->pmgp->n_iz), &(thee->pmgp->n_ipc),
00169     &(thee->pmgp->nwk), &(thee->pmgp->nwk)
00170   );
00171 }
00172
00173 /* We need some additional storage if: nonlinear & newton OR cgmgs */
00174 /* SMPBE Added - nonlin = 2 added since it mimics NPBE */
00175 if ( ( (thee->pmgp->nonlin == NONLIN_NPBE) || (thee->pmgp->nonlin == NONLIN_SMPBE) )
00176   && (thee->pmgp->meth == VSOL_Newton) ) || (thee->pmgp->meth == VSOL_CGMG) )
00177 {
00178   thee->pmgp->nwk += (2*(thee->pmgp->nf));
00179 }
00180
00181 Vnm_print(0, "Vpmg_ctor2: PMG chose nx = %d, ny = %d, nz = %d\n",
00182   thee->pmgp->nx, thee->pmgp->ny, thee->pmgp->nz);
00183 Vnm_print(0, "Vpmg_ctor2: PMG chose nlev = %d\n",
00184   thee->pmgp->nlev);
00185 Vnm_print(0, "Vpmg_ctor2: PMG chose nxc = %d, nyc = %d, nzc = %d\n",
00186   thee->pmgp->nxc, thee->pmgp->nyc, thee->pmgp->nzc);
00187 Vnm_print(0, "Vpmg_ctor2: PMG chose nf = %d, nc = %d\n",
00188   thee->pmgp->nf, thee->pmgp->nc);
00189 Vnm_print(0, "Vpmg_ctor2: PMG chose narr = %d, narrc = %d\n",
00190   thee->pmgp->narr, thee->pmgp->narrc);
00191 Vnm_print(0, "Vpmg_ctor2: PMG chose n_rpc = %d, n_iz = %d, n_ipc = %d\n",
00192   thee->pmgp->n_rpc, thee->pmgp->n_iz, thee->pmgp->n_ipc);
00193 Vnm_print(0, "Vpmg_ctor2: PMG chose nwk = %d, niwk = %d\n",
00194   thee->pmgp->nwk, thee->pmgp->niwk);
00195
00196 /* Allocate boundary storage */
00197 thee->gxfc = (double *)Vmem_malloc(thee->vmem,
00198   10*(thee->pmgp->ny)*(thee->pmgp->nz), sizeof(double));
00199 thee->gycf = (double *)Vmem_malloc(thee->vmem,
00200   10*(thee->pmgp->nx)*(thee->pmgp->nz), sizeof(double));
00201 thee->gzcf = (double *)Vmem_malloc(thee->vmem,
00202   10*(thee->pmgp->nx)*(thee->pmgp->ny), sizeof(double));
00203
00204 /* Warn users if they are using BCFL_MAP that
00205   we do not include external energies */
00206 if (thee->pmgp->bclf == BCFL_MAP)
00207   Vnm_print(2, "Vpmg_ctor2: \nWarning: External energies are not used in BCFL_MAP
calculations!\n");
00208
00209 if (focusFlag) {
00210   /* Overwrite any default or user-specified boundary condition

```

```

00211     * arguments; we are now committed to a calculation via focusing */
00212     if (thee->pmgp->bcfl != BCFL_FOCUS) {
00213         Vnm_print(2,
00214             "Vpmg_ctor2: reset boundary condition flag to BCFL_FOCUS!\n");
00215         thee->pmgp->bcfl = BCFL_FOCUS;
00216     }
00217
00218     /* Fill boundaries */
00219     Vnm_print(0, "Vpmg_ctor2: Filling boundary with old solution!\n");
00220     focusFillBound(thee, pmgOLD);
00221
00222     /* Calculate energetic contributions from region outside focusing
00223      * domain */
00224     if (energyFlag != PCE_NO) {
00225
00226         if (mgparm->type == MCT_PARALLEL) {
00227
00228             for (j=0; j<3; j++) {
00229                 partMin[j] = mgparm->partDisjCenter[j]
00230                     - 0.5*mgparm->partDisjLength[j];
00231                 partMax[j] = mgparm->partDisjCenter[j]
00232                     + 0.5*mgparm->partDisjLength[j];
00233             }
00234
00235         } else {
00236             for (j=0; j<3; j++) {
00237                 partMin[j] = mgparm->center[j] - 0.5*mgparm->glen[j];
00238                 partMax[j] = mgparm->center[j] + 0.5*mgparm->glen[j];
00239             }
00240         }
00241         extEnergy(thee, pmgOLD, energyFlag, partMin, partMax,
00242             mgparm->partDisjOwnSide);
00243     }
00244
00245 } else {
00246
00247     /* Ignore external energy contributions */
00248     thee->extQmEnergy = 0;
00249     thee->extDiEnergy = 0;
00250     thee->extQfEnergy = 0;
00251 }
00252
00253 /*
00254  * TODO: Move the dtor out of here. The current ctor is done in routines.c,
00255  * This was originally moved out to kill a memory leak. The dtor has
00256  * has been removed from initMG and placed back here to keep memory
00257  * usage low. killMG has been modified accordingly.
00258 */
00259 Vpmg_dtor(&pmgOLD);
00260
00261 /* Allocate partition vector storage */
00262 thee->pvec = (double *)Vmem_malloc(thee->vmem,
00263                                         (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->pmgp->nz), sizeof(double));
00264
00265 /* Allocate remaining storage */
00266 thee->iparm = (int *)Vmem_malloc(thee->vmem, 100, sizeof(int));
00267 thee->rparm = (double *)Vmem_malloc(thee->vmem, 100, sizeof(double));

```

```

00268 thee->iwork = (int *)Vmem_malloc(thee->vmem, thee->pmgp->niwk,
00269           sizeof(int));
00270 thee->rwork = (double *)Vmem_malloc(thee->vmem, thee->pmgp->nwk,
00271           sizeof(double));
00272 thee->charge = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00273           sizeof(double));
00274 thee->kappa = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00275           sizeof(double));
00276 thee->pot = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00277           sizeof(double));
00278 thee->epsx = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00279           sizeof(double));
00280 thee->epsy = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00281           sizeof(double));
00282 thee->epsz = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00283           sizeof(double));
00284 thee->alcf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00285           sizeof(double));
00286 thee->a2cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00287           sizeof(double));
00288 thee->a3cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00289           sizeof(double));
00290 thee->ccf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00291           sizeof(double));
00292 thee->fcf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00293           sizeof(double));
00294 thee->tcf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00295           sizeof(double));
00296 thee->u = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00297           sizeof(double));
00298 thee->xf = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nx),
00299           sizeof(double));
00300 thee->yf = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->ny),
00301           sizeof(double));
00302 thee->zf = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nz),
00303           sizeof(double));
00304
00305 /* Plop some of the parameters into the iparm and rparm arrays */
00306 F77PACKMG(thee->iparm, thee->rpParm, &(thee->pmgp->nwk), &(thee->pmgp->niwk),
00307   &(thee->pmgp->nx), &(thee->pmgp->ny), &(thee->pmgp->nz),
00308   &(thee->pmgp->nlev), &(thee->pmgp->nul), &(thee->pmgp->nu2),
00309   &(thee->pmgp->mgkey), &(thee->pmgp->itmax), &(thee->pmgp->istop),
00310   &(thee->pmgp->ipcon), &(thee->pmgp->nonlin), &(thee->pmgp->mgsmo),
00311   &(thee->pmgp->mgprol), &(thee->pmgp->mgcoar), &(thee->pmgp->mgsolv),
00312   &(thee->pmgp->mgdisc), &(thee->pmgp->iinfo), &(thee->pmgp->ertol),
00313   &(thee->pmgp->ipkey), &(thee->pmgp->omegal), &(thee->pmgp->omegan),
00314   &(thee->pmgp->irite), &(thee->pmgp->iperf));
00315
00316
00317 /* Initialize ion concentrations and valencies in PMG routines */
00318 zkappa2 = Vpbe_getZkappa2(thee->pbe);
00319 ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
00320 if (ionstr > 0.0) zks2 = 0.5/ionstr;
00321 else zks2 = 0.0;
00322 Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
00323
00324 /* Currently for SMPBE type calculations we do not want to apply a scale

```

```

00325     factor to the ionConc */
00326     switch(pmgp->ipkey){
00327         case IPKEY_SMPBE:
00328             F77MYPDEFINITSMPBE(&nion, ionQ, ionConc, &pbe->smvolume,&pbe->smsize);
00329             break;
00330         case IPKEY_NPBE:
00331             /* Else adjust the inoConc by scaling factor zks2 */
00332             for (i=0; i<nion; i++) ionConc[i] = zks2 * ionConc[i];
00333             F77MYPDEFINITNPBE(&nion, ionQ, ionConc);
00334             break;
00335         case IPKEY_LPBE:
00336             /* Else adjust the inoConc by scaling factor zks2 */
00337             for (i=0; i<nion; i++) ionConc[i] = zks2 * ionConc[i];
00338             F77MYPDEFINITLPBE(&nion, ionQ, ionConc);
00339             break;
00340         default:
00341             /* Else adjust the inoConc by scaling factor zks2 */
00342             for (i=0; i<nion; i++) ionConc[i] = zks2 * ionConc[i];
00343             break;
00344     }
00345
00346     /* Set the default chargeSrc for 5th order splines */
00347     thee->chargeSrc = mgparm->chgs;
00348
00349     /* Turn off restriction of observable calculations to a specific
00350     * partition */
00351     Vpmg_unsetPart(thee);
00352
00353     /* The coefficient arrays have not been filled */
00354     thee->filled = 0;
00355
00356     return 1;
00357 }
00358
00359 VPUBLIC int Vpmg_solve(Vpmg *thee) {
00360
00361     int i, nx, ny, nz, n;
00362     double zkappa2;
00363
00364     nx = thee->pmgp->nx;
00365     ny = thee->pmgp->ny;
00366     nz = thee->pmgp->nz;
00367     n = nx*ny*nz;
00368
00369     if (!(thee->filled)) {
00370         Vnm_print(2, "Vpmg_solve: Need to call Vpmg_fillco()!\n");
00371         return 0;
00372     }
00373
00374     /* Fill the "true solution" array */
00375     for (i=0; i<n; i++) {
00376         thee->tcf[i] = 0.0;
00377     }
00378
00379     /* Fill the RHS array */
00380     for (i=0; i<n; i++) {
00381         thee->fcf[i] = thee->charge[i];

```

```

00382     }
00383
00384     /* Fill the operator coefficient array. */
00385     for (i=0; i<n; i++) {
00386         thee->a1cf[i] = thee->epsx[i];
00387         thee->a2cf[i] = thee->epsy[i];
00388         thee->a3cf[i] = thee->epsz[i];
00389     }
00390
00391     /* Fill the nonlinear coefficient array by multiplying the kappa
00392      * accessibility array (containing values between 0 and 1) by zkappa2. */
00393     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00394     if (zkappa2 > VPMGSMALL) {
00395         for (i=0; i<n; i++) {
00396             thee->ccf[i] = zkappa2*thee->kappa[i];
00397         }
00398     } else {
00399         for (i=0; i<n; i++) {
00400             thee->ccf[i] = 0.0;
00401         }
00402     }
00403
00404     switch(thee->pmgp->meth) {
00405         /* CGMG (linear) */
00406         case VSOL_CGMG:
00407             F77CGMGDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00408             thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00409             thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00410             thee->fcf, thee->tcf);
00411             break;
00412         /* Newton (nonlinear) */
00413         case VSOL_Newton:
00414             F77NEWDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00415             thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00416             thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00417             thee->fcf, thee->tcf);
00418             break;
00419         /* MG (linear/nonlinear) */
00420         case VSOL_MG:
00421 #if 1
00422             F77MGDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00423             thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00424             thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00425             thee->fcf, thee->tcf);
00426 #else
00427             mgdrivc(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00428             thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00429             thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00430             thee->fcf, thee->tcf);
00431 #endif
00432             break;
00433         /* CGHS (linear/nonlinear) */
00434         case VSOL_CG:
00435             F77NCGHSDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00436             thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00437             thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00438             thee->fcf, thee->tcf);

```

```

00439         break;
00440     /* SOR (linear/nonlinear) */
00441     case VSOL_SOR:
00442     F77NSORDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00443         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00444         thee->gzcf, thee->alcf, thee->a2cf, thee->a3cf, thee->ccf,
00445         thee->fcaf, thee->tcf);
00446         break;
00447     /* GSRR (linear/nonlinear) */
00448     case VSOL_RBGS:
00449     F77NGSRBDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00450         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00451         thee->gzcf, thee->alcf, thee->a2cf, thee->a3cf, thee->ccf,
00452         thee->fcaf, thee->tcf);
00453         break;
00454     /* WJAC (linear/nonlinear) */
00455     case VSOL_WJ:
00456     F77NWJACDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00457         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00458         thee->gzcf, thee->alcf, thee->a2cf, thee->a3cf, thee->ccf,
00459         thee->fcaf, thee->tcf);
00460         break;
00461     /* RICH (linear/nonlinear) */
00462     case VSOL_Richardson:
00463     F77NRICHDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00464         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00465         thee->gzcf, thee->alcf, thee->a2cf, thee->a3cf, thee->ccf,
00466         thee->fcaf, thee->tcf);
00467         break;
00468     /* CGMG (linear) TEMPORARY USEAQUA */
00469     case VSOL_CGMGAqua:
00470         F77CGMGDRIVAAQUA(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00471             thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00472             thee->gzcf, thee->alcf, thee->a2cf, thee->a3cf, thee->ccf,
00473             thee->fcaf);
00474         break;
00475     /* Newton (nonlinear) TEMPORARY USEAQUA */
00476     case VSOL_NewtonAqua:
00477         F77NEWDRIVAAQUA(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00478             thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00479             thee->gzcf, thee->alcf, thee->a2cf, thee->a3cf, thee->ccf,
00480             thee->fcaf);
00481         break;
00482     /* Error handling */
00483     default:
00484         Vnm_print(2, "Vpmg_solve: invalid solver method key (%d)\n",
00485             thee->pmgp->key);
00486         return 0;
00487         break;
00488     }
00489
00490     return 1;
00491
00492 }
00493
00494
00495 VPUBLIC void Vpmg_dtor(Vpmg **thee) {

```

```

00496
00497     if ((*thee) != VNULL) {
00498         Vpmg_dtor2(*thee);
00499         Vmem_free(VNULL, 1, sizeof(Vpmg), (void **)thee);
00500         (*thee) = VNULL;
00501     }
00502
00503 }
00504
00505 VPUBLIC void Vpmg_dtor2(Vpmg *thee) {
00506
00507     /* Clear out the FORTRAN arrays */
00508     F77MYPDEFCLEAR();
00509
00510     /* Clean up the storage */
00511     Vmem_free(thee->vmem, 100, sizeof(int), (void **)(&(thee->iparm)));
00512     Vmem_free(thee->vmem, 100, sizeof(double), (void **)(&(thee->rparm)));
00513     Vmem_free(thee->vmem, thee->pmgp->nwk, sizeof(int),
00514             (void **)(&(thee->iwork)));
00515     Vmem_free(thee->vmem, thee->pmgp->nrwk, sizeof(double),
00516             (void **)(&(thee->rwork)));
00517     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00518             (void **)(&(thee->charge)));
00519     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00520             (void **)(&(thee->kappa)));
00521     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00522             (void **)(&(thee->pot)));
00523     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00524             (void **)(&(thee->epsx)));
00525     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00526             (void **)(&(thee->epsy)));
00527     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00528             (void **)(&(thee->epsz)));
00529     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00530             (void **)(&(thee->alcf)));
00531     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00532             (void **)(&(thee->a2cf)));
00533     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00534             (void **)(&(thee->a3cf)));
00535     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00536             (void **)(&(thee->ccf)));
00537     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00538             (void **)(&(thee->fcf)));
00539     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00540             (void **)(&(thee->tcf)));
00541     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00542             (void **)(&(thee->u)));
00543     Vmem_free(thee->vmem, 5*(thee->pmgp->nx), sizeof(double),
00544             (void **)(&(thee->xf)));
00545     Vmem_free(thee->vmem, 5*(thee->pmgp->ny), sizeof(double),
00546             (void **)(&(thee->yf)));
00547     Vmem_free(thee->vmem, 5*(thee->pmgp->nz), sizeof(double),
00548             (void **)(&(thee->zf)));
00549     Vmem_free(thee->vmem, 10*(thee->pmgp->ny)*(thee->pmgp->nz), sizeof(double),
00550             (void **)(&(thee->gxfc)));
00551     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->nz), sizeof(double),
00552             (void **)(&(thee->gycf)));

```

```

00553     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->ny), sizeof(double),
00554         (void **)(&(thee->gzcf)));
00555     Vmem_free(thee->vmem, (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->pmgp->nz),
00556         sizeof(double), (void **)(&(thee->pvec)));
00557
00558     Vmem_dtor(&(thee->vmem));
00559 }
00560
00561 VPUBLIC void Vpmg_setPart(Vpmg *thee, double lowerCorner[3],
00562     double upperCorner[3], int bflags[6]) {
00563
00564     Valist *alist;
00565     Vatom *atom;
00566     int i, j, k, nx, ny, nz;
00567     double xmin, ymin, zmin, x, y, z, hx, hy, hzed, xok, yok, zok;
00568     double x0,x1,y0,y1,z0,z1;
00569
00570     nx = thee->pmgp->nx;
00571     ny = thee->pmgp->ny;
00572     nz = thee->pmgp->nz;
00573     hx = thee->pmgp->hx;
00574     hy = thee->pmgp->hy;
00575     hzed = thee->pmgp->hzed;
00576     xmin = thee->pmgp->xcent - 0.5*hx*(nx-1);
00577     ymin = thee->pmgp->ycent - 0.5*hy*(ny-1);
00578     zmin = thee->pmgp->zcent - 0.5*hzed*(nz-1);
00579
00580     xok = 0;
00581     yok = 0;
00582     zok = 0;
00583
00584     /* We need have called Vpmg_fillco first */
00585
00586     alist = thee->pbe->alist;
00587
00588     Vnm_print(0, "Vpmg_setPart: lower corner = (%g, %g, %g)\n",
00589             lowerCorner[0], lowerCorner[1], lowerCorner[2]);
00590     Vnm_print(0, "Vpmg_setPart: upper corner = (%g, %g, %g)\n",
00591             upperCorner[0], upperCorner[1], upperCorner[2]);
00592     Vnm_print(0, "Vpmg_setPart: actual minima = (%g, %g, %g)\n",
00593             xmin, ymin, zmin);
00594     Vnm_print(0, "Vpmg_setPart: actual maxima = (%g, %g, %g)\n",
00595             xmin+hx*(nx-1), ymin+hy*(ny-1), zmin+hzed*(nz-1));
00596     Vnm_print(0, "Vpmg_setPart: bflag[FRONT] = %d\n",
00597             bflags[VAPBS_FRONT]);
00598     Vnm_print(0, "Vpmg_setPart: bflag[BACK] = %d\n",
00599             bflags[VAPBS_BACK]);
00600     Vnm_print(0, "Vpmg_setPart: bflag[LEFT] = %d\n",
00601             bflags[VAPBS_LEFT]);
00602     Vnm_print(0, "Vpmg_setPart: bflag[RIGHT] = %d\n",
00603             bflags[VAPBS_RIGHT]);
00604     Vnm_print(0, "Vpmg_setPart: bflag[UP] = %d\n",
00605             bflags[VAPBS_UP]);
00606     Vnm_print(0, "Vpmg_setPart: bflag[DOWN] = %d\n",
00607             bflags[VAPBS_DOWN]);
00608
00609     /* Identify atoms as inside, outside, or on the border

```

```

00610     If on the border, use the bflags to determine if there
00611     is an adjacent processor - if so, this atom should be equally
00612     shared. */
00613
00614     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00615         atom = Valist_getAtom(alist, i);
00616
00617         if ((atom->position[0] < upperCorner[0]) &&
00618             (atom->position[0] > lowerCorner[0])) xok = 1;
00619         else {
00620             if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL) &&
00621                 (bflags[VAPBS_LEFT] == 0)) xok = 1;
00622             else if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL) &&
00623                 (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00624             else if ((VABS(atom->position[0] - upperCorner[0]) < VPMGSMALL) &&
00625                 (bflags[VAPBS_RIGHT] == 0)) xok = 1;
00626             else if ((VABS(atom->position[0] - upperCorner[0]) < VPMGSMALL) &&
00627                 (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00628             else xok = 0;
00629         }
00630         if ((atom->position[1] < upperCorner[1]) &&
00631             (atom->position[1] > lowerCorner[1])) yok = 1;
00632         else {
00633             if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL) &&
00634                 (bflags[VAPBS_BACK] == 0)) yok = 1;
00635             else if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL) &&
00636                 (bflags[VAPBS_BACK] == 1)) yok = 0.5;
00637             else if ((VABS(atom->position[1] - upperCorner[1]) < VPMGSMALL) &&
00638                 (bflags[VAPBS_FRONT] == 0)) yok = 1;
00639             else if ((VABS(atom->position[1] - upperCorner[1]) < VPMGSMALL) &&
00640                 (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
00641             else yok = 0;
00642         }
00643         if ((atom->position[2] < upperCorner[2]) &&
00644             (atom->position[2] > lowerCorner[2])) zok = 1;
00645         else {
00646             if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL) &&
00647                 (bflags[VAPBS_DOWN] == 0)) zok = 1;
00648             else if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL) &&
00649                 (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00650             else if ((VABS(atom->position[2] - upperCorner[2]) < VPMGSMALL) &&
00651                 (bflags[VAPBS_UP] == 0)) zok = 1;
00652             else if ((VABS(atom->position[2] - upperCorner[2]) < VPMGSMALL) &&
00653                 (bflags[VAPBS_UP] == 1)) zok = 0.5;
00654             else zok = 0;
00655         }
00656
00657         atom->partID = xok*yok*zok;
00658     /*
00659     Vnm_print(1, "DEBUG (%s, %d): atom->position[0] - upperCorner[0] = %g\n",
00660             __FILE__, __LINE__, atom->position[0] - upperCorner[0]);
00661     Vnm_print(1, "DEBUG (%s, %d): atom->position[0] - lowerCorner[0] = %g\n",
00662             __FILE__, __LINE__, atom->position[0] - lowerCorner[0]);
00663     Vnm_print(1, "DEBUG (%s, %d): atom->position[1] - upperCorner[1] = %g\n",
00664             __FILE__, __LINE__, atom->position[1] - upperCorner[1]);
00665     Vnm_print(1, "DEBUG (%s, %d): atom->position[1] - lowerCorner[1] = %g\n",
00666             __FILE__, __LINE__, atom->position[1] - lowerCorner[1]);

```

```

00667 Vnm_print(1, "DEBUG (%s, %d): atom->position[2] - upperCorner[2] = %g\n",
00668     __FILE__, __LINE__, atom->position[2] - upperCorner[2]);
00669 Vnm_print(1, "DEBUG (%s, %d): atom->position[2] - lowerCorner[0] = %g\n",
00670     __FILE__, __LINE__, atom->position[2] - lowerCorner[2]);
00671 Vnm_print(1, "DEBUG (%s, %d): xok = %g, yok = %g, zok = %g\n",
00672     __FILE__, __LINE__, xok, yok, zok);
00673 */
00674
00675 }
00676
00677 /* Load up pvec -
00678   For all points within h{axis}/2 of a border - use a gradient
00679   to determine the pvec weight.
00680   Points on the boundary depend on the presence of an adjacent
00681   processor. */
00682
00683 for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 0.0;
00684
00685 for (i=0; i<nx; i++) {
00686     xok = 0.0;
00687     x = i*hx + xmin;
00688     if ((x < (upperCorner[0]-hx/2)) &&
00689         (x > (lowerCorner[0]+hx/2)))
00690         ) xok = 1.0;
00691     else if ((VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00692             (bflags[VAPBS_LEFT] == 0)) xok = 1.0;
00693     else if ((VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00694             (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00695     else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
00696             (bflags[VAPBS_RIGHT] == 0)) xok = 1.0;
00697     else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
00698             (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00699     else if ((x > (upperCorner[0] + hx/2)) || (x < (lowerCorner[0] - hx/2)))
00700     xok = 0.0;
00701     else if ((x < (upperCorner[0] + hx/2)) || (x > (lowerCorner[0] - hx/2)))
00702     {
00703         x0 = VMAX2(x - hx/2, lowerCorner[0]);
00704         x1 = VMIN2(x + hx/2, upperCorner[0]);
00705         xok = VABS(x1-x0)/hx;
00706         if (xok < 0.0) {
00707             if (VABS(xok) < VPMGSMALL) xok = 0.0;
00708             else {
00709                 Vnm_print(2, "Vpmg_setPart: fell off x-interval (%1.12E)!\n"
00710                         xok);
00711                 VASSERT(0);
00712             }
00713             if (xok > 1.0) {
00714                 if (VABS(xok - 1.0) < VPMGSMALL) xok = 1.0;
00715                 else {
00716                     Vnm_print(2, "Vpmg_setPart: fell off x-interval (%1.12E)!\n"
00717                         xok);
00718                     VASSERT(0);
00719                 }
00720             }
00721         }
00722     }
00723 }

```

```

00720         }
00721     } else yok = 0.0;
00722
00723     for (j=0; j<ny; j++) {
00724         yok = 0.0;
00725         y = j*hy + ymin;
00726         if ((y < (upperCorner[1]-hy/2)) && (y > (lowerCorner[1]+hy/2))) yok =
00727             1.0;
00728         else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00729                     (bflags[VAPBS_BACK] == 0)) yok = 1.0;
00730         else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00731                     (bflags[VAPBS_BACK] == 1)) yok = 0.5;
00732         else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00733                     (bflags[VAPBS_FRONT] == 0)) yok = 1.0;
00734         else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00735                     (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
00736         else if ((y > (upperCorner[1] + hy/2)) || (y < (lowerCorner[1] - hy/2
00737             ))) yok=0.0;
00738         else if ((y < (upperCorner[1] + hy/2)) || (y > (lowerCorner[1] - hy/2
00739             ))) {
00740             y0 = VMAX2(y - hy/2, lowerCorner[1]);
00741             y1 = VMIN2(y + hy/2, upperCorner[1]);
00742             yok = VABS(y1-y0)/hy;
00743             if (yok < 0.0) {
00744                 if (VABS(yok) < VPMGSMALL) yok = 0.0;
00745                 else {
00746                     Vnm_print(2, "Vpmg_setPart: fell off y-interval (%1.12E
00747 !\n",
00748                     yok);
00749                     VASSERT(0);
00750                 }
00751             if (yok > 1.0) {
00752                 if (VABS(yok - 1.0) < VPMGSMALL) yok = 1.0;
00753                 else {
00754                     Vnm_print(2, "Vpmg_setPart: fell off y-interval (%1.12E
00755 !\n",
00756                     yok);
00757                     VASSERT(0);
00758                 }
00759             else yok=0.0;
00760
00761             for (k=0; k<nz; k++) {
00762                 zok = 0.0;
00763                 z = k*hzed + zmin;
00764                 if ((z < (upperCorner[2]-hzed/2)) && (z > (lowerCorner[2]+hzed/2
00765             ))) zok = 1.0;
00766                 else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00767                     (bflags[VAPBS_DOWN] == 0)) zok = 1.0;
00768                 else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00769                     (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00770                 else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00771                     (bflags[VAPBS_UP] == 0)) zok = 1.0;

```

```

00771             else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00772                         (bflags[VAPBS_UP] == 1)) zok = 0.5;
00773             else if ((z > (upperCorner[2] + hzed/2)) || (z < (lowerCorner[2]
00774 - hzed/2))) zok=0.0;
00775             else if ((z < (upperCorner[2] + hzed/2)) || (z > (lowerCorner[2]
00776 - hzed/2))) {
00777                 z0 = VMAX2(z - hzed/2, lowerCorner[2]);
00778                 z1 = VMIN2(z + hzed/2, upperCorner[2]);
00779                 zok = VABS(z1-z0)/hzed;
00780
00781                 if (zok < 0.0) {
00782                     if (VABS(zok) < VPMGSMALL) zok = 0.0;
00783                     else {
00784                         Vnm_print(2, "Vpmg_setPart: fell off z-interval (%1.
00785 12E)!\\n",
00786                         zok);
00787                         VASSERT(0);
00788                     }
00789                 if (zok > 1.0) {
00790                     if (VABS(zok - 1.0) < VPMGSMALL) zok = 1.0;
00791                     else {
00792                         Vnm_print(2, "Vpmg_setPart: fell off z-interval (%1.
00793 12E)!\\n",
00794                         zok);
00795                         VASSERT(0);
00796                     }
00797                 }
00798             if (VABS(xok*yok*zok) < VPMGSMALL) thee->pvec[IJK(i,j,k)] = 0.0;
00799             else thee->pvec[IJK(i,j,k)] = xok*yok*zok;
00800
00801         }
00802     }
00803 }
00804 }
00805
00806 VPUBLIC void Vpmg_unsetPart(Vpmg *thee) {
00807
00808     int i, nx, ny, nz;
00809     Vatom *atom;
00810     Valist *alist;
00811
00812     VASSERT(thee != VNULL);
00813
00814     nx = thee->pmgp->nx;
00815     ny = thee->pmgp->ny;
00816     nz = thee->pmgp->nz;
00817     alist = thee->pbe->alist;
00818
00819     for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 1;
00820     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00821         atom = Valist_getAtom(alist, i);
00822         atom->partID = 1;
00823     }

```

```

00824 }
00825
00826 VPUBLIC int Vpmg_fillArray(Vpmg *thee, double *vec, Vdata_Type type,
00827     double parm, Vhal_PBEType pbetype, PBEparm *pbeparm) {
00828
00829     Vacc *acc = VNULL;
00830     Vpbe *pbe = VNULL;
00831     Vgrid *grid = VNULL;
00832     Vatom *atoms = VNULL;
00833     Valist *alist = VNULL;
00834     double position[3], hx, hy, hzed, xmin, ymin, zmin;
00835     double grad[3], eps, epss, epss, zmagic;
00836     int i, j, k, l, nx, ny, nz, ichop;
00837
00838     pbe = thee->pbe;
00839     acc = Vpbe_getVacc(pbe);
00840     nx = thee->pmgp->nx;
00841     ny = thee->pmgp->ny;
00842     nz = thee->pmgp->nz;
00843     hx = thee->pmgp->hx;
00844     hy = thee->pmgp->hy;
00845     hzed = thee->pmgp->hzed;
00846     xmin = thee->pmgp->xmin;
00847     ymin = thee->pmgp->ymin;
00848     zmin = thee->pmgp->zmin;
00849     epss = Vpbe_getSoluteDiel(pbe);
00850     epss = Vpbe_getSolventDiel(pbe);
00851     zmagic = Vpbe_getZmagic(pbe);
00852
00853     if (!!(thee->filled)) {
00854         Vnm_print(2, "Vpmg_fillArray: need to call Vpmg_fillco first!\n");
00855         return 0;
00856     }
00857
00858     switch (type) {
00859
00860         case VDT_CHARGE:
00861
00862             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->charge[i]/zmagic;
00863             break;
00864
00865         case VDT_DIELX:
00866
00867             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsx[i];
00868             break;
00869
00870         case VDT_DIELY:
00871
00872             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsy[i];
00873             break;
00874
00875         case VDT_DIELZ:
00876
00877             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsz[i];
00878             break;
00879
00880         case VDT_KAPPA:

```

```

00881             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->kappa[i];
00882             break;
00883
00884         case VDT_POT:
00885
00886             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->u[i];
00887             break;
00888
00889         case VDT_ATOMPOT:
00890             alist = thee->pbe->alist;
00891             atoms = alist[pbeparm->molid-1].atoms;
00892             grid = Vgrid_ctor(nx, ny, nz, hx, hy,
00893                             hzed, xmin, ymin, zmin, thee->u);
00894             for (i=0; i<alist[pbeparm->molid-1].number; i++) {
00895                 position[0] = atoms[i].position[0];
00896                 position[1] = atoms[i].position[1];
00897                 position[2] = atoms[i].position[2];
00898
00899                 Vgrid_value(grid, position, &vec[i]);
00900             }
00901             Vgrid_dtor(&grid);
00902             break;
00903
00904         case VDT_SMOL:
00905
00906             for (k=0; k<nz; k++) {
00907                 for (j=0; j<ny; j++) {
00908                     for (i=0; i<nx; i++) {
00909
00910                         position[0] = i*hx + xmin;
00911                         position[1] = j*hy + ymin;
00912                         position[2] = k*hzed + zmin;
00913
00914                         vec[IJK(i, j, k)] = (Vacc_molAcc(acc, position, parm));
00915                     }
00916                 }
00917             }
00918             break;
00919
00920         case VDT_SSPL:
00921
00922             for (k=0; k<nz; k++) {
00923                 for (j=0; j<ny; j++) {
00924                     for (i=0; i<nx; i++) {
00925
00926                         position[0] = i*hx + xmin;
00927                         position[1] = j*hy + ymin;
00928                         position[2] = k*hzed + zmin;
00929
00930                         vec[IJK(i, j, k)] = Vacc_splineAcc(acc, position, parm, 0);
00931                     }
00932                 }
00933             }
00934             break;
00935
00936         case VDT_VDW:
00937

```

```

00938
00939     for (k=0; k<nz; k++) {
00940         for (j=0; j<ny; j++) {
00941             for (i=0; i<nx; i++) {
00942
00943                 position[0] = i*hx + xmin;
00944                 position[1] = j*hy + ymin;
00945                 position[2] = k*hzed + zmin;
00946
00947                 vec[IJK(i,j,k)] = Vacc_vdwAcc(acc,position);
00948             }
00949         }
00950     }
00951     break;
00952
00953 case VDT_IVDW:
00954
00955     for (k=0; k<nz; k++) {
00956         for (j=0; j<ny; j++) {
00957             for (i=0; i<nx; i++) {
00958
00959                 position[0] = i*hx + xmin;
00960                 position[1] = j*hy + ymin;
00961                 position[2] = k*hzed + zmin;
00962
00963                 vec[IJK(i,j,k)] = Vacc_ivdwAcc(acc,position,parm);
00964             }
00965         }
00966     }
00967     break;
00968
00969 case VDT_LAP:
00970
00971     grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
00972                     theee->u);
00973     for (k=0; k<nz; k++) {
00974         for (j=0; j<ny; j++) {
00975             for (i=0; i<nx; i++) {
00976
00977                 if ((k==0) || (k==(nz-1)) ||
00978                     (j==0) || (j==(ny-1)) ||
00979                     (i==0) || (i==(nx-1))) {
00980
00981                     vec[IJK(i,j,k)] = 0;
00982
00983                 } else {
00984                     position[0] = i*hx + xmin;
00985                     position[1] = j*hy + ymin;
00986                     position[2] = k*hzed + zmin;
00987                     VASSERT(Vgrid_curvature(grid,position, 1,
00988                                     &(vec[IJK(i,j,k)])));
00989                 }
00990             }
00991         }
00992     }
00993     Vgrid_dtor(&grid);
00994     break;

```

```

00995
00996     case VDT_EDENS:
00997
00998         grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
00999             thee->u);
01000         for (k=0; k<nz; k++) {
01001             for (j=0; j<ny; j++) {
01002                 for (i=0; i<nx; i++) {
01003
01004                     position[0] = i*hx + xmin;
01005                     position[1] = j*hy + ymin;
01006                     position[2] = k*hzed + zmin;
01007                     VASSERT(Vgrid_gradient(grid, position, grad));
01008                     eps = epss + (epss-epsp)*Vacc_molAcc(acc, position,
01009                         pbe->solventRadius);
01010                     vec[IJK(i,j,k)] = 0.0;
01011                     for (l=0; l<3; l++)
01012                         vec[IJK(i,j,k)] += eps*VSQR(grad[l]);
01013                 }
01014             }
01015         }
01016         Vgrid_dtor(&grid);
01017         break;
01018
01019     case VDT_NDENS:
01020
01021         for (k=0; k<nz; k++) {
01022             for (j=0; j<ny; j++) {
01023                 for (i=0; i<nx; i++) {
01024
01025                     position[0] = i*hx + xmin;
01026                     position[1] = j*hy + ymin;
01027                     position[2] = k*hzed + zmin;
01028                     vec[IJK(i,j,k)] = 0.0;
01029                     if ( VABS(Vacc_ivdwAcc(acc,
01030                         position, pbe->maxIonRadius) - 1.0) < VSMALL) {
01031                         for (l=0; l<pbe->numIon; l++) {
01032                             if (pbetype == PBE_NPBE || pbetype == PBE_SMPBE /
01033 * SMPBE Added */
01034                         vec[IJK(i,j,k)] += (pbe->ionConc[l]
01035                             * Vcap_exp(-pbe->ionQ[l]*thee->u[IJK(i,j,
01036                                     k)],
01037                                     &ichop));
01038                     } else if (pbetype == PBE_LPBE){
01039                         vec[IJK(i,j,k)] += (pbe->ionConc[l]
01040                             * (1 - pbe->ionQ[l]*thee->u[IJK(i,j,k)]));
01041
01042                     }
01043                 }
01044             }
01045         break;
01046     case VDT_QDENS:
01047
01048

```

```

01049     for (k=0; k<nz; k++) {
01050         for (j=0; j<ny; j++) {
01051             for (i=0; i<nx; i++) {
01052
01053                 position[0] = i*hx + xmin;
01054                 position[1] = j*hy + ymin;
01055                 position[2] = k*hzed + zmin;
01056                 vec[IJK(i,j,k)] = 0.0;
01057                 if ( VABS(Vacc_ivdwAcc(acc,
01058                                 position, pbe->maxIonRadius) - 1.0) < VSMALL) {
01059                     for (l=0; l<pbe->numIon; l++) {
01060                         if (pbetype == PBE_NPBE || pbetype == PBE_SMPBE ||
01061 * SMPBE Added */
01062                         vec[IJK(i,j,k)] += (pbe->ionConc[l]
01063                         * pbe->ionQ[l]
01064                         * Vcap_exp(-pbe->ionQ[l]*thee->u[IJK(i,j,
01065                                         k)],
01066                                         &ichop));
01067                     } else if (pbetype == PBE_LPBE) {
01068                         vec[IJK(i,j,k)] += (pbe->ionConc[l]
01069                         * pbe->ionQ[l]
01070                         * (1 - pbe->ionQ[l]*thee->u[IJK(i,j,k)]));
01071                     }
01072                 }
01073             }
01074         }
01075         break;
01076     default:
01077
01078         Vnm_print(2, "main: Bogus data type (%d)!\n", type);
01079         return 0;
01080         break;
01081
01082     }
01083
01084     return 1;
01085 }
01086
01087 }
01088
01089 VPRIVATE double Vpmg_polarizEnergy(Vpmg *thee, int extFlag) {
01090
01091     int i, j, k, ijk, nx, ny, nz, iatom;
01092     double xmin, ymin, zmin, x, y, z, hx, hy, hzed, epsp, lap, pt[3];
01093     double T, pre, polq, dist2, dist, energy, q;
01094     double *charge, *pos, eps_w;
01095     Vgrid *potgrid;
01096     Vpbe *pbe;
01097     Valist *alist;
01098     Vatom *atom;
01099
01100     xmin = thee->pmgp->xmin;
01101     ymin = thee->pmgp->ymin;
01102     zmin = thee->pmgp->zmin;

```

```

01103     hx = thee->pmgp->hx;
01104     hy = thee->pmgp->hy;
01105     hzed = thee->pmgp->hzed;
01106     nx = thee->pmgp->nx;
01107     ny = thee->pmgp->ny;
01108     nz = thee->pmgp->nz;
01109     pbe = thee->pbe;
01110     epsp = Vpbe_getSoluteDiel(pbe);
01111     eps_w = Vpbe_getSolventDiel(pbe);
01112     alist = pbe->alist;
01113     charge = thee->charge;
01114
01115     /* Calculate the prefactor for Coulombic calculations */
01116     T = Vpbe_getTemperature(pbe);
01117     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
01118     pre = pre*(1.0e10);
01119
01120     /* Set up Vgrid object with solution */
01121     potgrid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin, thee->u);
01122
01123     /* Calculate polarization charge */
01124     energy = 0.0;
01125     for (i=1; i<(nx-1); i++) {
01126         pt[0] = xmin + hx*i;
01127         for (j=1; j<(ny-1); j++) {
01128             pt[1] = ymin + hy*j;
01129             for (k=1; k<(nz-1); k++) {
01130                 pt[2] = zmin + hzed*k;
01131
01132                 /* Calculate polarization charge */
01133                 VASSERT(Vgrid_curvature(potgrid, pt, 1, &lap));
01134                 ijk = IJK(i, j, k);
01135                 polq = charge[ijk] + epsp*lap*3.0;
01136
01137                 /* Calculate interaction energy with atoms */
01138                 if (VABS(polq) > VSMALL) {
01139                     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01140                         atom = Valist_getAtom(alist, iatom);
01141                         q = Vatom_getCharge(atom);
01142                         pos = Vatom_getPosition(atom);
01143                         dist2 = VSQR(pos[0]-pt[0]) + VSQR(pos[1]-pt[1]) \
01144                                 + VSQR(pos[2]-pt[2]);
01145                         dist = VSQRT(dist2);
01146
01147                         if (dist < VSMALL) {
01148                             Vnm_print(2, "Vpmg_polarizEnergy: atom on grid point
; ignoring!\n");
01149                         } else {
01150                             energy = energy + polq*q/dist;
01151                         }
01152                     }
01153                 }
01154             }
01155         }
01156     }
01157
01158     return pre*energy;

```

```

01159 }
01160
01161 VPUBLIC double Vpmg_energy(Vpmg *thee, int extFlag) {
01162
01163     double totEnergy = 0.0;
01164     double dielEnergy = 0.0;
01165     double qmEnergy = 0.0;
01166     double qfEnergy = 0.0;
01167
01168     VASSERT(thee != VNULL);
01169
01170     if ((thee->pmgp->nonlin) && (Vpbe_getBulkIonicStrength(thee->pbe) > 0.)) {
01171         Vnm_print(0, "Vpmg_energy: calculating full PBE energy\n");
01172         qmEnergy = Vpmg_qmEnergy(thee, extFlag);
01173         Vnm_print(0, "Vpmg_energy: qmEnergy = %1.12E kT\n", qmEnergy);
01174         qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01175         Vnm_print(0, "Vpmg_energy: qfEnergy = %1.12E kT\n", qfEnergy);
01176         dielEnergy = Vpmg_dielEnergy(thee, extFlag);
01177         Vnm_print(0, "Vpmg_energy: dielEnergy = %1.12E kT\n", dielEnergy);
01178         totEnergy = qfEnergy - dielEnergy - qmEnergy;
01179     } else {
01180         Vnm_print(0, "Vpmg_energy: calculating only q-phi energy\n");
01181         qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01182         Vnm_print(0, "Vpmg_energy: qfEnergy = %1.12E kT\n", qfEnergy);
01183         totEnergy = 0.5*qfEnergy;
01184     }
01185
01186     return totEnergy;
01187
01188 }
01189
01190 VPUBLIC double Vpmg_dielEnergy(Vpmg *thee, int extFlag) {
01191
01192     double hx, hy, hzed, energy, nrgx, nrgy, nrgz, pvecx, pvecy, pvecz;
01193     int i, j, k, nx, ny, nz;
01194
01195     VASSERT(thee != VNULL);
01196
01197     /* Get the mesh information */
01198     nx = thee->pmgp->nx;
01199     ny = thee->pmgp->ny;
01200     nz = thee->pmgp->nz;
01201     hx = thee->pmgp->hx;
01202     hy = thee->pmgp->hy;
01203     hzed = thee->pmgp->hzed;
01204
01205     energy = 0.0;
01206
01207     if (!thee->filled) {
01208         Vnm_print(2, "Vpmg_dielEnergy: Need to call Vpmg_fillco!\n");
01209         VASSERT(0);
01210     }
01211
01212     for (k=0; k<(nz-1); k++) {
01213         for (j=0; j<(ny-1); j++) {
01214             for (i=0; i<(nx-1); i++) {
01215                 pvecx = 0.5*(thee->pvec[IJK(i, j, k)]+thee->pvec[IJK(i+1, j, k)]);

```

```

01216         pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j+1,k)]);
01217         pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k+1)]);
01218         nrgx = thee->epsx[IJK(i,j,k)]*pvecx
01219             * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i+1,j,k)])/hx);
01220         nrgy = thee->epsy[IJK(i,j,k)]*pvecy
01221             * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j+1,k)])/hy);
01222         nrgz = thee->epsz[IJK(i,j,k)]*pvecz
01223             * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j,k+1)])/hzed);
01224         energy += (nrgx + nrgy + nrgz);
01225     }
01226 }
01227 }
01228
01229 energy = 0.5*energy*hx*hy*hzed;
01230 energy = energy/Vpbe_getZmagic(thee->pbe);
01231
01232 if (extFlag == 1) energy += (thee->extDiEnergy);
01233
01234 return energy;
01235 }
01236
01237 VPUBLIC double Vpmg_dielGradNorm(Vpmg *thee) {
01238
01239     double hx, hy, hzed, energy, nrgx, nrgy, nrgz, pvecx, pvecy, pvecz;
01240     int i, j, k, nx, ny, nz;
01241
01242     VASSERT(thee != VNNULL);
01243
01244     /* Get the mesh information */
01245     nx = thee->pmgp->nx;
01246     ny = thee->pmgp->ny;
01247     nz = thee->pmgp->nz;
01248     hx = thee->pmgp->hx;
01249     hy = thee->pmgp->hy;
01250     hzed = thee->pmgp->hzed;
01251
01252     energy = 0.0;
01253
01254     if (!thee->filled) {
01255         Vnm_print(2, "Vpmg_dielGradNorm: Need to call Vpmg_fillco!\n");
01256         VASSERT(0);
01257     }
01258
01259     for (k=1; k<nz; k++) {
01260         for (j=1; j<ny; j++) {
01261             for (i=1; i<nx; i++) {
01262                 pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i-1,j,k)]);
01263                 pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j-1,k)]);
01264                 pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k-1)]);
01265                 nrgx = pvecx
01266                     * VSQR((thee->epsx[IJK(i,j,k)]-thee->epsx[IJK(i-1,j,k)])/hx);
01267                 nrgy = pvecy
01268                     * VSQR((thee->epsy[IJK(i,j,k)]-thee->epsy[IJK(i,j-1,k)])/hy);
01269                 nrgz = pvecz
01270                     * VSQR((thee->epsz[IJK(i,j,k)]-thee->epsz[IJK(i,j,k-1)])/hzed);
01271                 energy += VSQRT(nrgx + nrgy + nrgz);
01272             }
01273         }
01274     }

```

```

01273         }
01274     }
01275
01276     energy = energy*hx*hy*hzed;
01277
01278     return energy;
01279 }
01280
01281 VPUBLIC double Vpmg_qmEnergy(Vpmg *thee, int extFlag) {
01282
01283     double energy;
01284
01285     if (thee->pbe->ipkey == IPKEY_SMPBE) {
01286         energy = Vpmg_qmEnergySMPBE(thee,extFlag);
01287     }else{
01288         energy = Vpmg_qmEnergyNONLIN(thee,extFlag);
01289     }
01290
01291     return energy;
01292 }
01293
01294 VPRIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee, int extFlag) {
01295
01296     double hx, hy, hzed, energy, ionConc[MAXION], ionRadii[MAXION];
01297     double ionQ[MAXION], zkappa2, ionstr, zks2;
01298     int i, j, nx, ny, nz, nion, ichop, nchop;
01299
01300     VASSERT(thee != VNULL);
01301
01302     /* Get the mesh information */
01303     nx = thee->pmgp->nx;
01304     ny = thee->pmgp->ny;
01305     nz = thee->pmgp->nz;
01306     hx = thee->pmgp->hx;
01307     hy = thee->pmgp->hy;
01308     hzed = thee->pmgp->hzed;
01309     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01310     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01311
01312     /* Bail if we're at zero ionic strength */
01313     if (zkappa2 < VSMALL) {
01314
01315 #ifndef VAPBSQUIET
01316         Vnm_print(0, "Vpmg_qmEnergy: Zero energy for zero ionic strength!\n");
01317 #endif
01318
01319     return 0.0;
01320 }
01321     zks2 = 0.5*zkappa2/ionstr;
01322
01323     if (!thee->filled) {
01324         Vnm_print(2, "Vpmg_qmEnergy: Need to call Vpmg_fillco()!\n");
01325         VASSERT(0);
01326     }
01327
01328     energy = 0.0;
01329     nchop = 0;

```

```

01330     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
01331     if (thee->pmgp->nonlin) {
01332         Vnm_print(0, "Vpmg_qmEnergy: Calculating nonlinear energy\n");
01333         for (i=0; i<(nx*ny*nz); i++) {
01334             if (thee->pvec[i]*thee->kappa[i] > VSMALL) {
01335                 for (j=0; j<nion; j++) {
01336                     energy += (thee->pvec[i]*thee->kappa[i]*zks2
01337                         * ionConc[j]
01338                         * (Vcap_exp(-ionQ[j]*thee->u[i], &ichop)-1.0));
01339                     nchop += ichop;
01340                 }
01341             }
01342         }
01343         if (nchop > 0){
01344             Vnm_print(2, "Vpmg_qmEnergy: Chopped EXP %d times!\n",nchop);
01345             Vnm_print(2, "\nERROR! Detected large potential values in energy evaluation!
01346 \nERROR! This calculation failed -- please report to the APBS developers!\n\n");
01347         }
01348     } else {
01349         /* Zkappa2 OK here b/c LPBE approx */
01350         Vnm_print(0, "Vpmg_qmEnergy: Calculating linear energy\n");
01351         for (i=0; i<(nx*ny*nz); i++) {
01352             if (thee->pvec[i]*thee->kappa[i] > VSMALL)
01353                 energy += (thee->pvec[i]*zkappa2*thee->kappa[i]*VSQR(thee->u[i]));
01354         }
01355         energy = 0.5*energy;
01356     }
01357     energy = energy*hx*hy*hzed;
01358     energy = energy/Vpbe_getZmagic(thee->pbe);
01359
01360     if (extFlag == 1) energy += thee->extQmEnergy;
01361
01362     return energy;
01363 }
01364
01365 VPUBLIC double Vpmg_qmEnergySMPBE(Vpmg *thee, int extFlag) {
01366
01367     double hx, hy, hzed, energy, ionConc[MAXION], ionRadii[MAXION];
01368     double ionQ[MAXION], zkappa2, ionstr, zks2;
01369     int i, j, nx, ny, nz, nion, ichop, nchop;
01370
01371     /* SMPB Modification (vchu, 09/21/06)*/
01372     /* variable declarations for SMPB energy terms */
01373     double a, k, z1, z2, z3, cb1, cb2, cb3;
01374     double a1, a2, a3, c1, c2, c3, currEnergy;
01375     double fracOccA, fracOccB, fracOccC, phi, gpark, denom, Na;
01376     int ichop1, ichop2, ichop3;
01377
01378     VASSERT(thee != VNULL);
01379
01380     /* Get the mesh information */
01381     nx = thee->pmgp->nx;
01382     ny = thee->pmgp->ny;
01383     nz = thee->pmgp->nz;
01384     hx = thee->pmgp->hx;

```

```

01385     hy = thee->pmgp->hy;
01386     hzed = thee->pmgp->hzed;
01387     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01388     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01389
01390     /* Bail if we're at zero ionic strength */
01391     if (zkappa2 < VSMALL) {
01392
01393 #ifndef VAPBSQUIET
01394         Vnm_print(0, "Vpmg_qmEnergySMPBE: Zero energy for zero ionic strength!\n"
01395     );
01396 #endif
01397     return 0.0;
01398 }
01399     zks2 = 0.5*zkappa2/ionstr;
01400
01401     if (!thee->filled) {
01402         Vnm_print(2, "Vpmg_qmEnergySMPBE: Need to call Vpmg_fillco() !\n");
01403         VASSERT(0);
01404     }
01405
01406     energy = 0.0;
01407     nchop = 0;
01408     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
01409
01410     /* SMPB Modification (vchu, 09/21/06) */
01411     /* Extensive modification to the first part of the if statement
01412 where that handles the thee->pmgp->nonlin part. Basically, I've
01413 deleted all of the original code and written my own code that computes
01414 the electrostatic free energy in the SMPB framework. Definitely really hacky
01415 at this stage of the game, but gets the job done. The second part of the
01416 if statement (the part that handles linear poisson-boltzmann) has been deleted
01417 because there will be no linearized SMPB energy.. */
01418
01419     z1 = ionQ[0];
01420     z2 = ionQ[1];
01421     z3 = ionQ[2];
01422     cb1 = ionConc[0];
01423     cb2 = ionConc[1];
01424     cb3 = ionConc[2];
01425     a = thee->pbe->smvolume;
01426     k = thee->pbe->smsize;
01427     Na = 6.022045000e-04; /* Converts from Molar to N/A^3 */
01428
01429     fracOccA = Na*cb1*VCUB(a);
01430     fracOccB = Na*cb2*VCUB(a);
01431     fracOccC = Na*cb3*VCUB(a);
01432
01433     phi = (fracOccA/k) + fracOccB + fracOccC;
01434
01435     if (thee->pmgp->nonlin) {
01436         Vnm_print(0, "Vpmg_qmEnergySMPBE: Calculating nonlinear energy using SMP
01437 B functional!\n");
01438         for (i=0; i<(nx*ny*nz); i++) {
01439             if (((k-1) > VSMALL) && (thee->pvec[i]*thee->kappa[i] > VSMALL)) {
01439

```

```

01440     a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01441     a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01442     a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01443
01444     nchop += ichop1 + ichop2 + ichop3;
01445
01446     gpark = (1 - phi + (fracOccA/k)*a1);
01447     denom = VPOW(gpark, k) + VPOW(1-fracOccB-fracOccC, k-1)*(fracOccB*a2+fracOccC
* a3);
01448
01449     if (cb1 > VSMALL) {
01450         c1 = Na*cb1*VPOW(gpark, k-1)*a1/denom;
01451         if(c1 != c1) c1 = 0.;
01452     } else c1 = 0.;
01453
01454     if (cb2 > VSMALL) {
01455         c2 = Na*cb2*VPOW(1-fracOccB-fracOccC, k-1)*a2/denom;
01456         if(c2 != c2) c2 = 0.;
01457     } else c2 = 0.;
01458
01459     if (cb3 > VSMALL) {
01460         c3 = Na*cb3*VPOW(1-fracOccB-fracOccC, k-1)*a3/denom;
01461         if(c3 != c3) c3 = 0.;
01462     } else c3 = 0.;
01463
01464     currEnergy = k*VLOG((1-(c1*VCUB(a)/k)-c2*VCUB(a)-c3*VCUB(a))/(1-phi))
01465     -(k-1)*VLOG((1-c2*VCUB(a)-c3*VCUB(a))/(1-phi+(fracOccA/k)));
01466
01467     energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01468
01469 } else if (thee->pvec[i]*thee->kappa[i] > VSMALL) {
01470
01471     a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01472     a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01473     a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01474
01475     nchop += ichop1 + ichop2 + ichop3;
01476
01477     gpark = (1 - phi + (fracOccA)*a1);
01478     denom = gpark + (fracOccB*a2+fracOccC*a3);
01479
01480     if (cb1 > VSMALL) {
01481         c1 = Na*cb1*a1/denom;
01482         if(c1 != c1) c1 = 0.;
01483     } else c1 = 0.;
01484
01485     if (cb2 > VSMALL) {
01486         c2 = Na*cb2*a2/denom;
01487         if(c2 != c2) c2 = 0.;
01488     } else c2 = 0.;
01489
01490     if (cb3 > VSMALL) {
01491         c3 = Na*cb3*a3/denom;
01492         if(c3 != c3) c3 = 0.;
01493     } else c3 = 0.;
01494
01495     currEnergy = VLOG((1-c1*VCUB(a)-c2*VCUB(a)-c3*VCUB(a))/(1-fracOccA-fracOccB-f

```

```

        racOccC));
01496     energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01498 }
01499 }
01500
01501     energy = -energy/VCUB(a);
01502
01503     if (nchop > 0) Vnm_print(2, "Vpmg_qmEnergySMPBE: Chopped EXP %d times!\n",
01504     ", nchop);
01505
01506 } else {
01507     /* Zkappa2 OK here b/c LPBE approx */
01508     Vnm_print(0, "Vpmg_qmEnergySMPBE: ERROR: NO LINEAR ENERGY!! Returning 0!
01509 \n");
01510     energy = 0.0;
01511 }
01512
01513     energy = energy*hx*hy*hzed;
01514
01515     if (extFlag == 1) energy += thee->extQmEnergy;
01516
01517     return energy;
01518 }
01519
01520 VPUBLIC double Vpmg_qfEnergy(Vpmg *thee, int extFlag) {
01521     double energy = 0.0;
01523
01524     VASSERT(thee != VNULL);
01525
01526     if ((thee->useChargeMap) || (thee->chargeMeth == VCM_BSPL2)) {
01527         energy = Vpmg_qfEnergyVolume(thee, extFlag);
01528     } else {
01529         energy = Vpmg_qfEnergyPoint(thee, extFlag);
01530     }
01531
01532     return energy;
01533 }
01534
01535 VPRIVATE double Vpmg_qfEnergyPoint(Vpmg *thee, int extFlag) {
01536
01537     int iatom, nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01538     double xmax, ymax, zmax, xmin, ymin, zmin, hx, hy, hzed, ifloat, jfloat;
01539     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01540     double *u;
01541     double *pvec;
01542     Valist *alist;
01543     Vatom *atom;
01544     Vpbe *pbe;
01545
01546     pbe = thee->pbe;
01547     alist = pbe->alist;
01548     VASSERT(alist != VNULL);
01549

```

```

01550     /* Get the mesh information */
01551     nx = thee->pmgp->nx;
01552     ny = thee->pmgp->ny;
01553     nz = thee->pmgp->nz;
01554     hx = thee->pmgp->hx;
01555     hy = thee->pmgp->hy;
01556     hzed = thee->pmgp->hzed;
01557     xmax = thee->pmgp->xmax;
01558     ymax = thee->pmgp->ymax;
01559     zmax = thee->pmgp->zmax;
01560     xmin = thee->pmgp->xmin;
01561     ymin = thee->pmgp->ymin;
01562     zmin = thee->pmgp->zmin;
01563
01564     u = thee->u;
01565     pvec = thee->pvec;
01566
01567     energy = 0.0;
01568
01569     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01570
01571         /* Get atomic information */
01572         atom = Valist_getAtom(alist, iatom);
01573
01574         position = Vatom_getPosition(atom);
01575         charge = Vatom_getCharge(atom);
01576
01577         /* Figure out which vertices we're next to */
01578         ifloat = (position[0] - xmin)/hx;
01579         jfloat = (position[1] - ymin)/hy;
01580         kfloat = (position[2] - zmin)/hzed;
01581         ihi = (int)ceil(ifloat);
01582         ilo = (int)floor(ifloat);
01583         jhi = (int)ceil(jfloat);
01584         jlo = (int)floor(jfloat);
01585         khi = (int)ceil(kfloat);
01586         klo = (int)floor(kfloat);
01587
01588         if (atom->partID > 0) {
01589
01590             if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01591                 (ilo>=0) && (jlo>=0) && (klo>=0)) {
01592
01593                 /* Now get trilinear interpolation constants */
01594                 dx = ifloat - (double)(ilo);
01595                 dy = jfloat - (double)(jlo);
01596                 dz = kfloat - (double)(klo);
01597                 uval =
01598                     dx*dy*dz*u[IJK(ihi,jhi,khi)]
01599                     + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
01600                     + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
01601                     + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
01602                     + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
01603                     + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
01604                     + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
01605                     + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
01606                 energy += (uval*charge*atom->partID);

```

```

01607         } else if (thee->pmgp->bclf != BCFL_FOCUS) {
01608             Vnm_print(2, "Vpmg_qfEnergy: Atom #%d at (%4.3f, %4.3f, \
01609 %4.3f) is off the mesh (ignoring)!\\n",
01610                     iatom, position[0], position[1], position[2]);
01611             }
01612         }
01613     }
01614
01615     if (extFlag) energy += thee->extQfEnergy;
01616
01617     return energy;
01618 }
01619
01620 VPUBLIC double Vpmg_qfAtomEnergy(Vpmg *thee, Vatom *atom) {
01621
01622     int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01623     double xmax, xmin, ymax, ymin, zmax, zmin, hx, hy, hzed, ifloat, jfloat;
01624     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01625     double *u;
01626
01627
01628     /* Get the mesh information */
01629     nx = thee->pmgp->nx;
01630     ny = thee->pmgp->ny;
01631     nz = thee->pmgp->nz;
01632     hx = thee->pmgp->hx;
01633     hy = thee->pmgp->hy;
01634     hzed = thee->pmgp->hzed;
01635     xmax = thee->xf[nx-1];
01636     ymax = thee->yf[ny-1];
01637     zmax = thee->zf[nz-1];
01638     xmin = thee->xf[0];
01639     ymin = thee->yf[0];
01640     zmin = thee->zf[0];
01641
01642     u = thee->u;
01643
01644     energy = 0.0;
01645
01646
01647     position = Vatom_getPosition(atom);
01648     charge = Vatom_getCharge(atom);
01649
01650     /* Figure out which vertices we're next to */
01651     ifloat = (position[0] - xmin)/hx;
01652     jfloat = (position[1] - ymin)/hy;
01653     kfloat = (position[2] - zmin)/hzed;
01654     ihi = (int)ceil(ifloat);
01655     ilo = (int)floor(ifloat);
01656     jhi = (int)ceil(jfloat);
01657     jlo = (int)floor(jfloat);
01658     khi = (int)ceil(kfloat);
01659     klo = (int)floor(kfloat);
01660
01661     if (atom->partID > 0) {
01662         if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01663

```

```

01664             (ilo>=0) && (jlo>=0) && (klo>=0)) {
01665
01666         /* Now get trilinear interpolation constants */
01667         dx = ifloat - (double)(ilo);
01668         dy = jfloat - (double)(jlo);
01669         dz = kfloat - (double)(klo);
01670         uval =
01671             dix*dy*dz*u[IJK(ihi,jhi,khi)]
01672             + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
01673             + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
01674             + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
01675             + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
01676             + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
01677             + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
01678             + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
01679         energy += (uval*charge*atom->partID);
01680     } else if (thee->pmgp->bclf != BCFL_FOCUS) {
01681         Vnm_print(2, "Vpmg_qfAtomEnergy: Atom at (%4.3f, %4.3f, \
01682 %4.3f) is off the mesh (ignoring)!\\n",
01683             position[0], position[1], position[2]);
01684     }
01685 }
01686
01687     return energy;
01688 }
01689
01690 VPRIvATE double Vpmg_qfEnergyVolume (Vpmg *thee, int extFlag) {
01691
01692     double hx, hy, hzed, energy;
01693     int i, nx, ny, nz;
01694
01695     VASSERT(thee != VNULL);
01696
01697     /* Get the mesh information */
01698     nx = thee->pmgp->nx;
01699     ny = thee->pmgp->ny;
01700     nz = thee->pmgp->nz;
01701     hx = thee->pmgp->hx;
01702     hy = thee->pmgp->hy;
01703     hzed = thee->pmgp->hzed;
01704
01705     if (!thee->filled) {
01706         Vnm_print(2, "Vpmg_qfEnergyVolume: need to call Vpmg_fillco!\\n");
01707         VASSERT(0);
01708     }
01709
01710     energy = 0.0;
01711     Vnm_print(0, "Vpmg_qfEnergyVolume: Calculating energy\\n");
01712     for (i=0; i<(nx*ny*nz); i++) {
01713         energy += (thee->pvec[i]*thee->u[i]*thee->charge[i]);
01714     }
01715     energy = energy*hx*hy*hzed/Vpbe_getZmagic(thee->pbe);
01716
01717     if (extFlag == 1) energy += thee->extQfEnergy;
01718
01719     return energy;
01720 }
```

```

01721
01722 VPRIVATE void Vpmg_splineSelect(int srfm,Vacc *acc,double *gpos,double win,
01723           double infrad,Vatom *atom,double *force) {
01724
01725     switch (srfm) {
01726         case VSM_SPLINE :
01727             Vacc_splineAccGradAtomNorm(acc, gpos, win, infrad, atom, force);
01728             break;
01729         case VSM_SPLINE3:
01730             Vacc_splineAccGradAtomNorm3(acc, gpos, win, infrad, atom, force);
01731             break;
01732         case VSM_SPLINE4 :
01733             Vacc_splineAccGradAtomNorm4(acc, gpos, win, infrad, atom, force);
01734             break;
01735         default:
01736             Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
01737             return;
01738     }
01739
01740     return;
01741 }
01742
01743 VPRIVATE void focusFillBound(Vpmg *thee, Vpmg *pmgOLD) {
01744
01745     Vpbe *pbe;
01746     double hxOLD, hyOLD, hzOLD, xminOLD, yminOLD, zminOLD, xmaxOLD, ymaxOLD;
01747     double zmaxOLD;
01748     int nxOLD, nyOLD, nzOLD;
01749     double hxNEW, hyNEW, hzNEW, xminNEW, yminNEW, zminNEW, xmaxNEW, ymaxNEW;
01750     double zmaxNEW;
01751     int nxNEW, nyNEW, nzNEW;
01752     int i, j, k, ihi, ilo, jhi, jlo, khi, klo, nx, ny, nz;
01753     double x, y, z, dx, dy, dz, ifloat, jfloat, kfloat, uval;
01754     double eps_w, T, prel, xkappa, size, *apos, charge, pos[3];
01755
01756     double uvalMin, uvalMax;
01757     double *data;
01758
01759     /* Calculate new problem dimensions */
01760     hxNEW = thee->pmgp->hx;
01761     hyNEW = thee->pmgp->hy;
01762     hzNEW = thee->pmgp->hzed;
01763     nx = thee->pmgp->nx;
01764     ny = thee->pmgp->ny;
01765     nz = thee->pmgp->nz;
01766     nxNEW = thee->pmgp->nx;
01767     nyNEW = thee->pmgp->ny;
01768     nzNEW = thee->pmgp->nz;
01769     xminNEW = thee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0;
01770     xmaxNEW = thee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0;
01771     yminNEW = thee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0;
01772     ymaxNEW = thee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0;
01773     zminNEW = thee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0;
01774     zmaxNEW = thee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0;
01775
01776     if (pmgOLD != VNULL) {
01777         /* Relevant old problem parameters */

```

```

01778     hxOLD = pmgOLD->pmgp->hx;
01779     hyOLD = pmgOLD->pmgp->hy;
01780     hzOLD = pmgOLD->pmgp->hzed;
01781     nxOLD = pmgOLD->pmgp->nx;
01782     nyOLD = pmgOLD->pmgp->ny;
01783     nzOLD = pmgOLD->pmgp->nz;
01784     xminOLD = pmgOLD->pmgp->xcent - ((double) (nxOLD-1)*hxOLD)/2.0;
01785     xmaxOLD = pmgOLD->pmgp->xcent + ((double) (nxOLD-1)*hxOLD)/2.0;
01786     yminOLD = pmgOLD->pmgp->ycent - ((double) (nyOLD-1)*hyOLD)/2.0;
01787     ymaxOLD = pmgOLD->pmgp->ycent + ((double) (nyOLD-1)*hyOLD)/2.0;
01788     zminOLD = pmgOLD->pmgp->zcent - ((double) (nzOLD-1)*hzOLD)/2.0;
01789     zmaxOLD = pmgOLD->pmgp->zcent + ((double) (nzOLD-1)*hzOLD)/2.0;
01790
01791     data = pmgOLD->u;
01792 }else{
01793 /* Relevant old problem parameters */
01794     hxOLD = thee->potMap->hx;
01795     hyOLD = thee->potMap->hy;
01796     hzOLD = thee->potMap->hzed;
01797     nxOLD = thee->potMap->nx;
01798     nyOLD = thee->potMap->ny;
01799     nzOLD = thee->potMap->nz;
01800     xminOLD = thee->potMap->xmin;
01801     xmaxOLD = thee->potMap->xmax;
01802     yminOLD = thee->potMap->ymin;
01803     ymaxOLD = thee->potMap->ymax;
01804     zminOLD = thee->potMap->zmin;
01805     zmaxOLD = thee->potMap->zmax;
01806
01807     data = thee->potMap->data;
01808 }
01809 /* BOUNDARY CONDITION SETUP FOR POINTS OFF OLD MESH:
01810 * For each "atom" (only one for bcfl=1), we use the following formula to
01811 * calculate the boundary conditions:
01812 *   g(x) = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_B T} *
01813 *           \frac{\exp(-xkappa*(d - a))}{1+xkappa*a}
01814 *           * 1/d
01815 * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
01816 * We only need to evaluate some of these prefactors once:
01817 *   prel = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_B T}
01818 * which gives the potential as
01819 *   g(x) = prel * q/d * \frac{\exp(-xkappa*(d - a))}{1+xkappa*a}
01820 */
01821     pbe = thee->pbe;
01822     eps_w = Vpbe_getSolventDiel(pbe); /* Dimensionless */
01823     T = Vpbe_getTemperature(pbe); /* K */
01824     prel = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
01825
01826 /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
01827 * m/A, then we will only need to deal with distances and sizes in
01828 * Angstroms rather than meters. */
01829     xkappa = Vpbe_getXkappa(pbe); /* A^{-1} */
01830     prel = prel*(1.0e10);
01831     size = Vpbe_getSoluteRadius(pbe);
01832     apos = Vpbe_getSoluteCenter(pbe);
01833     charge = Vunit_ec*Vpbe_getSoluteCharge(pbe);
01834

```

```

01835     /* Check for rounding error */
01836     if (VABS(xminOLD-xminNEW) < VSMALL) xminNEW = xminOLD;
01837     if (VABS(xmaxOLD-xmaxNEW) < VSMALL) xmaxNEW = xmaxOLD;
01838     if (VABS(yminOLD-yminNEW) < VSMALL) yminNEW = yminOLD;
01839     if (VABS(ymaxOLD-ymaxNEW) < VSMALL) ymaxNEW = ymaxOLD;
01840     if (VABS(zminOLD-zminNEW) < VSMALL) zminNEW = zminOLD;
01841     if (VABS(zmaxOLD-zmaxNEW) < VSMALL) zmaxNEW = zmaxOLD;
01842
01843
01844     /* Sanity check: make sure we're within the old mesh */
01845     Vnm_print(0, "VPMG::focusFillBound -- New mesh mins = %g, %g, %g\n",
01846               xminNEW, yminNEW, zminNEW);
01847     Vnm_print(0, "VPMG::focusFillBound -- New mesh maxs = %g, %g, %g\n",
01848               xmaxNEW, ymaxNEW, zmaxNEW);
01849     Vnm_print(0, "VPMG::focusFillBound -- Old mesh mins = %g, %g, %g\n",
01850               xminOLD, yminOLD, zminOLD);
01851     Vnm_print(0, "VPMG::focusFillBound -- Old mesh maxs = %g, %g, %g\n",
01852               xmaxOLD, ymaxOLD, zmaxOLD);
01853
01854     /* The following is obsolete; we'll substitute analytical boundary
01855      * condition values when the new mesh falls outside the old */
01856     if ((xmaxNEW>xmaxOLD) || (ymaxNEW>ymaxOLD) || (zmaxNEW>zmaxOLD) ||
01857         (xminOLD>xminNEW) || (yminOLD>yminNEW) || (zminOLD>zminNEW)) {
01858
01859         Vnm_print(2, "Vpmg::focusFillBound -- new mesh not contained in old!\n");
01860
01861         Vnm_print(2, "Vpmg::focusFillBound -- old mesh min = (%g, %g, %g)\n",
01862                   xminOLD, yminOLD, zminOLD);
01863         Vnm_print(2, "Vpmg::focusFillBound -- old mesh max = (%g, %g, %g)\n",
01864                   xmaxOLD, ymaxOLD, zmaxOLD);
01865         Vnm_print(2, "Vpmg::focusFillBound -- new mesh min = (%g, %g, %g)\n",
01866                   xminNEW, yminNEW, zminNEW);
01867         Vnm_print(2, "Vpmg::focusFillBound -- new mesh max = (%g, %g, %g)\n",
01868                   xmaxNEW, ymaxNEW, zmaxNEW);
01869         fflush(stderr);
01870         VASSERT(0);
01871     }
01872     uvalMin = VPMGSMALL;
01873     uvalMax = -VPMGSMALL;
01874
01875     /* Fill the "i" boundaries (dirichlet) */
01876     for (k=0; k<nzNEW; k++) {
01877         for (j=0; j<nyNEW; j++) {
01878             /* Low X face */
01879             x = xminNEW;
01880             y = yminNEW + j*hyNEW;
01881             z = zminNEW + k*hzNEW;
01882             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
01883             OLD-VSMALL)) &&
01884                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
01885             OLD+VSMALL))) {
01886                 ifloat = (x - xminOLD)/hxOLD;
01887                 jfloat = (y - yminOLD)/hyOLD;
01888                 kfload = (z - zminOLD)/hzOLD;
01889                 ihi = (int)ceil(ifloat);
01890                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;

```

```

01889         ilo = (int)floor(ifloat);
01890         if (ilo < 0) ilo = 0;
01891         jhi = (int)ceil(jfloat);
01892         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
01893         jlo = (int)floor(jfloat);
01894         if (jlo < 0) jlo = 0;
01895         khi = (int)ceil(kfloat);
01896         if (khi > (nzOLD-1)) khi = nzOLD-1;
01897         klo = (int)floor(kfloat);
01898         if (klo < 0) klo = 0;
01899         dx = ifloat - (double)(ilo);
01900         dy = jfloat - (double)(jlo);
01901         dz = kfloat - (double)(klo);
01902         nx = nxOLD; ny = nyOLD; nz = nzOLD;
01903         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
01904         + dx*dy*(1.0-dz)*(data[IJK(ihi,jlo,khi)])
01905         + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
01906         + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
01907         + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
01908         + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
01909         + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
01910         nx = nxNEW; ny = nyNEW; nz = nzNEW;
01911     } else {
01912     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
01913             %g!\n", __FILE__, __LINE__, x, y, z);
01914     Vnm_print(2, "focusfillBound (%s, %d): old mesh lower corner at \
01915             %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
01916     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
01917             %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
01918     VASSERT(0);
01919     }
01920     nx = nxNEW; ny = nyNEW; nz = nzNEW;
01921     thee->gxcf[IJKx(j,k,0)] = uval;
01922     if(uval < uvalMin) uvalMin = uval;
01923     if(uval > uvalMax) uvalMax = uval;
01924
01925     /* High X face */
01926     x = xmaxNEW;
01927     if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
01928             OLD-VSMALL)) && (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
01929             OLD+VSMALL))) {
01930         ifloat = (x - xminOLD)/hxOLD;
01931         jfloat = (y - yminOLD)/hyOLD;
01932         kfloat = (z - zminOLD)/hzOLD;
01933         ihi = (int)ceil(ifloat);
01934         if (ihi > (nxOLD-1)) ihi = nxOLD-1;
01935         ilo = (int)floor(ifloat);
01936         if (ilo < 0) ilo = 0;
01937         jhi = (int)ceil(jfloat);
01938         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
01939         jlo = (int)floor(jfloat);
01940         if (jlo < 0) jlo = 0;
01941         khi = (int)ceil(kfloat);
01942         if (khi > (nzOLD-1)) khi = nzOLD-1;

```

```

01943         klo = (int)floor(kfloat);
01944         if (klo < 0) klo = 0;
01945         dx = ifloat - (double)(ilo);
01946         dy = jfloat - (double)(jlo);
01947         dz = kfloat - (double)(klo);
01948         nx = nxOLD; ny = nyOLD; nz = nzOLD;
01949         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
01950         + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
01951         + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
01952         + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,khi)])
01953         + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
01954         + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
01955         + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
01956         + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
01957         nx = nxNEW; ny = nyNEW; nz = nzNEW;
01958     } else {
01959     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
01960             %g!\n", __FILE__, __LINE__, x, y, z);
01961     Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
01962             %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
01963     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
01964             %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
01965     VASSERT(0);
01966     }
01967     nx = nxNEW; ny = nyNEW; nz = nzNEW;
01968     thee->gxcf[IJKx(j,k,1)] = uval;
01969     if(uval < uvalMin) uvalMin = uval;
01970     if(uval > uvalMax) uvalMax = uval;
01971
01972     /* Zero Neumann conditions */
01973     nx = nxNEW; ny = nyNEW; nz = nzNEW;
01974     thee->gxcf[IJKx(j,k,2)] = 0.0;
01975     nx = nxNEW; ny = nyNEW; nz = nzNEW;
01976     thee->gxcf[IJKx(j,k,3)] = 0.0;
01977   }
01978 }
01979
01980 /* Fill the "j" boundaries (dirichlet) */
01981 for (k=0; k<nzNEW; k++) {
01982   for (i=0; i<nxNEW; i++) {
01983     /* Low Y face */
01984     x = xminNEW + i*hxNEW;
01985     y = yminNEW;
01986     z = zminNEW + k*hzNEW;
01987     if((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
01988           OLD-VSMALL)) &&
01989           (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
01990           OLD+VSMALL))) {
01991       ifloat = (x - xminOLD)/hxOLD;
01992       jfloat = (y - yminOLD)/hyOLD;
01993       kfloat = (z - zminOLD)/hzOLD;
01994       ihi = (int)ceil(ifloat);
01995       if (ihi > (nxOLD-1)) ihi = nxOLD-1;
01996       ilo = (int)floor(ifloat);
01997       if (ilo < 0) ilo = 0;
01998       jhi = (int)ceil(jfloat);

```

```

01997         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
01998         jlo = (int)floor(jffloat);
01999         if (jlo < 0) jlo = 0;
02000         khi = (int)ceil(kffloat);
02001         if (khi > (nzOLD-1)) khi = nzOLD-1;
02002         klo = (int)floor(kffloat);
02003         if (klo < 0) klo = 0;
02004         dx = ifloat - (double)(ilo);
02005         dy = jfloat - (double)(jlo);
02006         dz = kffloat - (double)(klo);
02007         nx = nxOLD; ny = nyOLD; nz = nzOLD;
02008         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02009         + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02010         + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02011         + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02012         + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02013         + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02014         + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02015         + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02016         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02017     } else {
02018     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02019             %g!\n", __FILE__, __LINE__, x, y, z);
02020     Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02021             %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02022     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02023             %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02024     VASSERT(0);
02025     }
02026     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02027     thee->gycf[IJKy(i,k,0)] = uval;
02028     if(uval < uvalMin) uvalMin = uval;
02029     if(uval > uvalMax) uvalMax = uval;
02030
02031     /* High Y face */
02032     y = ymaxNEW;
02033     if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
02034     OLD-VSMALL)) &&
02035             (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
02036     OLD+VSMALL))) {
02037         ifloat = (x - xminOLD)/hxOLD;
02038         jfloat = (y - yminOLD)/hyOLD;
02039         kffloat = (z - zminOLD)/hzOLD;
02040         ihi = (int)ceil(ifloat);
02041         if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02042         ilo = (int)floor(ifloat);
02043         if (ilo < 0) ilo = 0;
02044         jhi = (int)ceil(jffloat);
02045         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02046         jlo = (int)floor(jffloat);
02047         if (jlo < 0) jlo = 0;
02048         khi = (int)ceil(kffloat);
02049         if (khi > (nzOLD-1)) khi = nzOLD-1;
02050         klo = (int)floor(kffloat);
02051         if (klo < 0) klo = 0;
02052         dx = ifloat - (double)(ilo);

```

```

02051             dy = jfloat - (double)(jlo);
02052             dz = kfloat - (double)(klo);
02053             nx = nxOLD; ny = nyOLD; nz = nzOLD;
02054             uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02055             + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02056             + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02057             + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02058             + (1.0-dx)*dy*dz*(data[IJK(ihi,jhi,khi)])
02059             + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02060             + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02061             + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)]);
02062             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02063         } else {
02064             Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02065             %g!\n", __FILE__, __LINE__, x, y, z);
02066             Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02067             %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02068             Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02069             %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02070             VASSERT(0);
02071         }
02072         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02073         thee->gycf[IJKy(i,k,1)] = uval;
02074         if(uval < uvalMin) uvalMin = uval;
02075         if(uval > uvalMax) uvalMax = uval;
02076
02077         /* Zero Neumann conditions */
02078         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02079         thee->gycf[IJKy(i,k,2)] = 0.0;
02080         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02081         thee->gycf[IJKy(i,k,3)] = 0.0;
02082     }
02083 }
02084
02085 /* Fill the "k" boundaries (dirichlet) */
02086 for (j=0; j<nyNEW; j++) {
02087     for (i=0; i<nxNEW; i++) {
02088         /* Low Z face */
02089         x = xminNEW + i*hxNEW;
02090         y = yminNEW + j*hyNEW;
02091         z = zminNEW;
02092         if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
02093             OLD-VSMALL)) &&
02094             (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
02095             OLD+VSMALL))) {
02096             ifloat = (x - xminOLD)/hxOLD;
02097             jfloat = (y - yminOLD)/hyOLD;
02098             kfloat = (z - zminOLD)/hzOLD;
02099             ihi = (int)ceil(ifloat);
02100             if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02101             ilo = (int)floor(ifloat);
02102             if (ilo < 0) ilo = 0;
02103             jhi = (int)ceil(jfloat);
02104             if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02105             jlo = (int)floor(jfloat);
02106             if (jlo < 0) jlo = 0;

```

```

02105         khi = (int)ceil(kffloat);
02106         if (khi > (nzOLD-1)) khi = nzOLD-1;
02107         klo = (int)floor(kffloat);
02108         if (klo < 0) klo = 0;
02109         dx = ifloat - (double)(ilo);
02110         dy = jfloat - (double)(jlo);
02111         dz = kffloat - (double)(klo);
02112         nx = nxOLD; ny = nyOLD; nz = nzOLD;
02113         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02114         + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02115         + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02116         + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02117         + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02118         + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02119         + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02120         + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02121         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02122     } else {
02123     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02124     %g!\n", __FILE__, __LINE__, x, y, z);
02125     Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02126     %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02127     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02128     %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02129     VASSERT(0);
02130     }
02131     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02132     theee->gzcf[IJKz(i,j,0)] = uval;
02133     if(uval < uvalMin) uvalMin = uval;
02134     if(uval > uvalMax) uvalMax = uval;
02135
02136     /* High Z face */
02137     z = zmaxNEW;
02138     if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
02139     OLD-VSMALL)) && (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
02140     OLD+VSMALL))) {
02141         ifloat = (x - xminOLD)/hxOLD;
02142         jfloat = (y - yminOLD)/hyOLD;
02143         kffloat = (z - zminOLD)/hzOLD;
02144         ihi = (int)ceil(ifloat);
02145         if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02146         ilo = (int)floor(ifloat);
02147         if (ilo < 0) ilo = 0;
02148         jhi = (int)ceil(jfloat);
02149         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02150         jlo = (int)floor(jfloat);
02151         if (jlo < 0) jlo = 0;
02152         khi = (int)ceil(kffloat);
02153         if (khi > (nzOLD-1)) khi = nzOLD-1;
02154         klo = (int)floor(kffloat);
02155         if (klo < 0) klo = 0;
02156         dx = ifloat - (double)(ilo);
02157         dy = jfloat - (double)(jlo);
02158         dz = kffloat - (double)(klo);
02159         nx = nxOLD; ny = nyOLD; nz = nzOLD;

```

```

02159      uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02160      + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02161      + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02162      + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02163      + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02164      + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02165      + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02166      + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02167      nx = nxNEW; ny = nyNEW; nz = nzNEW;
02168      } else {
02169      Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02170      %g!\n", __FILE__, __LINE__, x, y, z);
02171      Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02172      %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02173      Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02174      %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02175      VASSERT(0);
02176      }
02177      nx = nxNEW; ny = nyNEW; nz = nzNEW;
02178      thee->gzcf[IJKz(i,j,1)] = uval;
02179      if(uval < uvalMin) uvalMin = uval;
02180      if(uval > uvalMax) uvalMax = uval;
02181
02182      /* Zero Neumann conditions */
02183      nx = nxNEW; ny = nyNEW; nz = nzNEW;
02184      thee->gzcf[IJKz(i,j,2)] = 0.0;
02185      nx = nxNEW; ny = nyNEW; nz = nzNEW;
02186      thee->gzcf[IJKz(i,j,3)] = 0.0;
02187
02188  }
02189
02190  if((uvalMin < SINH_MIN) || (uvalMax > SINH_MAX)){
02191  Vnm_print(2, "\nfocusFillBound: WARNING! Unusually large potential values\n" \
02192          "                                detected on the focusing boundary! \n" \
02193          "                                Convergence not guaranteed for NPBE/NRPBE calculation
02194  s!\n");
02195
02196  }
02197
02198 VPRIATE void extEnergy(Vpmg *thee, Vpmg *pmgOLD, PBEparm_calcEnergy extFlag,
02199      double partMin[3], double partMax[3], int bflags[6]) {
02200
02201      Vatom *atom;
02202      double hxNEW, hyNEW, hzNEW;
02203      double lowerCorner[3], upperCorner[3];
02204      int nxNEW, nyNEW, nzNEW;
02205      int nxOLD, nyOLD, nzOLD;
02206      int i,j,k;
02207      double xmin, xmax, ymin, ymax, zmin, zmax;
02208      double hxOLD, hyOLD, hzOLD;
02209      double xval, yval, zval;
02210      double x,y,z;
02211      int nx, ny, nz;
02212

```

```

02213 /* Set the new external energy contribution to zero. Any external
02214 * contributions from higher levels will be included in the appropriate
02215 * energy function call. */
02216 theee->extQmEnergy = 0;
02217 theee->extQfEnergy = 0;
02218 theee->extDiEnergy = 0;
02219
02220 /* New problem dimensions */
02221 hxNEW = theee->pmgp->hx;
02222 hyNEW = theee->pmgp->hy;
02223 hzNEW = theee->pmgp->hzed;
02224 nxNEW = theee->pmgp->nx;
02225 nyNEW = theee->pmgp->ny;
02226 nzNEW = theee->pmgp->nz;
02227 lowerCorner[0] = theee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0;
02228 upperCorner[0] = theee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0;
02229 lowerCorner[1] = theee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0;
02230 upperCorner[1] = theee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0;
02231 lowerCorner[2] = theee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0;
02232 upperCorner[2] = theee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0;
02233
02234 Vnm_print(0, "VPMG::extEnergy: energy flag = %d\n", extFlag);
02235
02236 /* Old problem dimensions */
02237 nxOLD = pmgOLD->pmgp->nx;
02238 nyOLD = pmgOLD->pmgp->ny;
02239 nzOLD = pmgOLD->pmgp->nz;
02240
02241 /* Create a partition based on the new problem dimensions */
02242 /* Vnm_print(1, "DEBUG (%s, %d): extEnergy calling Vpmg_setPart for old PMG.
02243 \n",
02244 __FILE__, __LINE__); */
02245 Vpmg_setPart(pmgOLD, lowerCorner, upperCorner, bflags);
02246
02247 Vnm_print(0, "VPMG::extEnergy: Finding extEnergy dimensions...\n");
02248 Vnm_print(0, "VPMG::extEnergy Disj part lower corner = (%g, %g, %g)\n",
02249 partMin[0], partMin[1], partMin[2]);
02250 Vnm_print(0, "VPMG::extEnergy Disj part upper corner = (%g, %g, %g)\n",
02251 partMax[0], partMax[1], partMax[2]);
02252
02253 /* Find the old dimensions */
02254
02255 hxOLD = pmgOLD->pmgp->hx;
02256 hyOLD = pmgOLD->pmgp->hy;
02257 hzOLD = pmgOLD->pmgp->hzed;
02258 xmin = pmgOLD->pmgp->xcent - 0.5*hxOLD*(nxOLD-1);
02259 ymin = pmgOLD->pmgp->ycent - 0.5*hyOLD*(nyOLD-1);
02260 zmin = pmgOLD->pmgp->zcent - 0.5*hzOLD*(nzOLD-1);
02261 xmax = xmin+hxOLD*(nxOLD-1);
02262 ymax = ymin+hyOLD*(nyOLD-1);
02263 zmax = zmin+hzOLD*(nzOLD-1);
02264
02265 Vnm_print(0, "VPMG::extEnergy Old lower corner = (%g, %g, %g)\n",
02266 xmin, ymin, zmin);
02267 Vnm_print(0, "VPMG::extEnergy Old upper corner = (%g, %g, %g)\n",
02268 xmax, ymax, zmax);

```

```

02269
02270     /* Flip the partition, but do not include any points that will
02271     be included by another processor */
02272
02273     nx = nxOLD;
02274     ny = nyOLD;
02275     nz = nzOLD;
02276
02277     for(i=0; i<nx; i++) {
02278         xval = 1;
02279         x = i*hxOLD + xmin;
02280         if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02281         else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;
02282
02283         for(j=0; j<ny; j++) {
02284             yval = 1;
02285             y = j*hyOLD + ymin;
02286             if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02287             else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;
02288
02289             for(k=0; k<nz; k++) {
02290                 zval = 1;
02291                 z = k*hzOLD + zmin;
02292                 if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
02293                 else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
02294
02295                 if (pmgOLD->pvec[IJK(i,j,k)] > VSMALL) pmgOLD->pvec[IJK(i,j,k)] =
1.0;
02296                 pmgOLD->pvec[IJK(i,j,k)] = (1 - (pmgOLD->pvec[IJK(i,j,k)])) * (xv
al*yval*zval);
02297             }
02298         }
02299     }
02300
02301     for (i=0; i<Valist_getNumberAtoms(thee->pbe->alist); i++) {
02302         xval=1;
02303         yval=1;
02304         zval=1;
02305         atom = Valist_getAtom(thee->pbe->alist, i);
02306         x = atom->position[0];
02307         y = atom->position[1];
02308         z = atom->position[2];
02309         if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02310         else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;
02311         if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02312         else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;
02313         if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
02314         else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
02315         if (atom->partID > VSMALL) atom->partID = 1.0;
02316         atom->partID = (1 - atom->partID) * (xval*yval*zval);
02317     }
02318
02319     /* Now calculate the energy on inverted subset of the domain */
02320     thee->extQmEnergy = Vpmg_qmEnergy(pmgOLD, 1);
02321     Vnm_print(0, "VPMG::extEnergy: extQmEnergy = %g kT\n", thee->extQmEnergy);
02322     thee->extQfEnergy = Vpmg_qfEnergy(pmgOLD, 1);
02323     Vnm_print(0, "VPMG::extEnergy: extQfEnergy = %g kT\n", thee->extQfEnergy);

```

```

02324     thee->extDiEnergy = Vpmg_dielEnergy(pmgOLD, 1);
02325     Vnm_print(0, "VPMG::extEnergy: extDiEnergy = %g kT\n", thee->extDiEnergy);
02326     Vpmg_unsetPart(pmgOLD);
02327 }
02328
02329 VPRIVATE double bcfl1sp(double size, double *apos, double charge,
02330     double xkappa, double prel, double *pos) {
02331
02332     double dist, val;
02333
02334     dist = VSQRT(VSQR(pos[0]-apos[0]) + VSQR(pos[1]-apos[1])
02335         + VSQR(pos[2]-apos[2]));
02336     if (xkappa > VSMALL) {
02337         val = prel*(charge/dist)*VEEXP(-xkappa*(dist-size))
02338         / (1+xkappa*size);
02339     } else {
02340         val = prel*(charge/dist);
02341     }
02342
02343     return val;
02344 }
02345
02346 VPRIVATE void bcfl1(double size, double *apos, double charge,
02347     double xkappa, double prel, double *gxcf, double *gycf, double *gzcf,
02348     double *xf, double *yf, double *zf, int nx, int ny, int nz) {
02349
02350     int i, j, k;
02351     double dist, val;
02352     double gpos[3];
02353
02354     /* the "i" boundaries (dirichlet) */
02355     for (k=0; k<nz; k++) {
02356         gpos[2] = zf[k];
02357         for (j=0; j<ny; j++) {
02358             gpos[1] = yf[j];
02359             gpos[0] = xf[0];
02360             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02361                 + VSQR(gpos[2]-apos[2]));
02362             if (xkappa > VSMALL) {
02363                 val = prel*(charge/dist)*VEEXP(-xkappa*(dist-size))
02364                 / (1+xkappa*size);
02365             } else {
02366                 val = prel*(charge/dist);
02367             }
02368             gxcf[IJKx(j,k,0)] += val;
02369             gpos[0] = xf[nx-1];
02370             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02371                 + VSQR(gpos[2]-apos[2]));
02372             if (xkappa > VSMALL) {
02373                 val = prel*(charge/dist)*VEEXP(-xkappa*(dist-size))
02374                 / (1+xkappa*size);
02375             } else {
02376                 val = prel*(charge/dist);
02377             }
02378             gxcf[IJKx(j,k,1)] += val;
02379         }
02380     }

```

```

02381
02382     /* the "j" boundaries (dirichlet) */
02383     for (k=0; k<nz; k++) {
02384         gpos[2] = zf[k];
02385         for (i=0; i<nx; i++) {
02386             gpos[0] = xf[i];
02387             gpos[1] = yf[0];
02388             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02389             + VSQR(gpos[2]-apos[2]));
02390             if (xkappa > VSMALL) {
02391                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02392             / (1+xkappa*size);
02393             } else {
02394                 val = pre1*(charge/dist);
02395             }
02396             gycf[IJKy(i,k,0)] += val;
02397             gpos[1] = yf[ny-1];
02398             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02399             + VSQR(gpos[2]-apos[2]));
02400             if (xkappa > VSMALL) {
02401                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02402             / (1+xkappa*size);
02403             } else {
02404                 val = pre1*(charge/dist);
02405             }
02406             gycf[IJKy(i,k,1)] += val;
02407         }
02408     }
02409
02410     /* the "k" boundaries (dirichlet) */
02411     for (j=0; j<ny; j++) {
02412         gpos[1] = yf[j];
02413         for (i=0; i<nx; i++) {
02414             gpos[0] = xf[i];
02415             gpos[2] = zf[0];
02416             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02417             + VSQR(gpos[2]-apos[2]));
02418             if (xkappa > VSMALL) {
02419                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02420             / (1+xkappa*size);
02421             } else {
02422                 val = pre1*(charge/dist);
02423             }
02424             gzcf[IJKz(i,j,0)] += val;
02425             gpos[2] = zf[nz-1];
02426             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02427             + VSQR(gpos[2]-apos[2]));
02428             if (xkappa > VSMALL) {
02429                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02430             / (1+xkappa*size);
02431             } else {
02432                 val = pre1*(charge/dist);
02433             }
02434             gzcf[IJKz(i,j,1)] += val;
02435         }
02436     }
02437 }
```

```
02438
02439 VPRIVATE void bcfl2(double size, double *apos,
02440                 double charge, double *dipole, double *quad,
02441                 double xkappa, double eps_p, double eps_w, double T,
02442                 double *gxcf, double *gycf, double *gzcf,
02443                 double *xf, double *yf, double *zf,
02444                 int nx, int ny, int nz) {
02445
02446     int i, j, k;
02447     double val;
02448     double gpos[3], tensor[3];
02449     double ux, uy, uz, xr, yr, zr;
02450     double qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
02451     double dist, pre;
02452
02453     VASSERT(dipole != VNULL);
02454     ux = dipole[0];
02455     uy = dipole[1];
02456     uz = dipole[2];
02457     if (quad != VNULL) {
02458         /* The factor of 1/3 results from using a
02459         traceless quadrupole definition. See, for example,
02460         "The Theory of Intermolecular Forces" by A.J. Stone,
02461         Chapter 3. */
02462     qxx = quad[0] / 3.0;
02463     qxy = quad[1] / 3.0;
02464     qxz = quad[2] / 3.0;
02465     qyx = quad[3] / 3.0;
02466     qyy = quad[4] / 3.0;
02467     qyz = quad[5] / 3.0;
02468     qzx = quad[6] / 3.0;
02469     qzy = quad[7] / 3.0;
02470     qzz = quad[8] / 3.0;
02471     } else {
02472     qxx = 0.0;
02473     qxy = 0.0;
02474     qxz = 0.0;
02475     qyx = 0.0;
02476     qyy = 0.0;
02477     qyz = 0.0;
02478     qzx = 0.0;
02479     qzy = 0.0;
02480     qzz = 0.0;
02481     }
02482
02483     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
02484     pre = pre*(1.0e10);
02485
02486     /* the "i" boundaries (dirichlet) */
02487     for (k=0; k<nz; k++) {
02488         gpos[2] = zf[k];
02489         for (j=0; j<ny; j++) {
02490             gpos[1] = yf[j];
02491             gpos[0] = xf[0];
02492             xr = gpos[0] - apos[0];
02493             yr = gpos[1] - apos[1];
02494             zr = gpos[2] - apos[2];
```

```

02495     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02496     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02497     val = pre*charge*tensor[0];
02498     val -= pre*ux*xr*tensor[1];
02499     val -= pre*uy*yr*tensor[1];
02500     val -= pre*uz*zr*tensor[1];
02501     val += pre*qxx*xr*xr*tensor[2];
02502     val += pre*qyy*yrr*yr*tensor[2];
02503     val += pre*qzz*zr*zr*tensor[2];
02504     val += pre*2.0*qxy*xr*yr*tensor[2];
02505     val += pre*2.0*qxz*xr*zr*tensor[2];
02506     val += pre*2.0*qyz*yr*zr*tensor[2];
02507     gxcf[IJKx(j,k,0)] += val;
02508
02509     gpos[0] = xf[nx-1];
02510     xr = gpos[0] - apos[0];
02511     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02512     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02513     val = pre*charge*tensor[0];
02514     val -= pre*ux*xr*tensor[1];
02515     val -= pre*uy*yr*tensor[1];
02516     val -= pre*uz*zr*tensor[1];
02517     val += pre*qxx*xr*xr*tensor[2];
02518     val += pre*qyy*yrr*yr*tensor[2];
02519     val += pre*qzz*zr*zr*tensor[2];
02520     val += pre*2.0*qxy*xr*yr*tensor[2];
02521     val += pre*2.0*qxz*xr*zr*tensor[2];
02522     val += pre*2.0*qyz*yr*zr*tensor[2];
02523     gxcf[IJKx(j,k,1)] += val;
02524 }
02525 }
02526
02527 /* the "j" boundaries (dirichlet) */
02528 for (k=0; k<nz; k++) {
02529     gpos[2] = zf[k];
02530     for (i=0; i<nx; i++) {
02531         gpos[0] = xf[i];
02532         gpos[1] = yf[0];
02533         xr = gpos[0] - apos[0];
02534         yr = gpos[1] - apos[1];
02535         zr = gpos[2] - apos[2];
02536         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02537         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02538         val = pre*charge*tensor[0];
02539         val -= pre*ux*xr*tensor[1];
02540         val -= pre*uy*yr*tensor[1];
02541         val -= pre*uz*zr*tensor[1];
02542         val += pre*qxx*xr*xr*tensor[2];
02543         val += pre*qyy*yrr*yr*tensor[2];
02544         val += pre*qzz*zr*zr*tensor[2];
02545         val += pre*2.0*qxy*xr*yr*tensor[2];
02546         val += pre*2.0*qxz*xr*zr*tensor[2];
02547         val += pre*2.0*qyz*yr*zr*tensor[2];
02548         gycf[IJKy(i,k,0)] += val;
02549
02550     gpos[1] = yf[ny-1];
02551     yr = gpos[1] - apos[1];

```

```

02552         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02553         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02554         val = pre*charge*tensor[0];
02555         val -= pre*ux*xr*tensor[1];
02556         val -= pre*uy*yr*tensor[1];
02557         val -= pre*uz*zr*tensor[1];
02558         val += pre*qxx*xr*xr*tensor[2];
02559         val += pre*qyy*yr*yr*tensor[2];
02560         val += pre*qzz*zr*zr*tensor[2];
02561         val += pre*2.0*qxy*xr*yr*tensor[2];
02562         val += pre*2.0*qxz*xr*zr*tensor[2];
02563         val += pre*2.0*qyz*yr*zr*tensor[2];
02564         gycf[IJKy(i,k,1)] += val;
02565     }
02566 }
02567
02568 /* the "k" boundaries (dirichlet) */
02569 for (j=0; j<ny; j++) {
02570     gpos[1] = yf[j];
02571     for (i=0; i<nx; i++) {
02572         gpos[0] = xf[i];
02573         gpos[2] = zf[0];
02574         xr = gpos[0] - apos[0];
02575         yr = gpos[1] - apos[1];
02576         zr = gpos[2] - apos[2];
02577         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02578         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02579         val = pre*charge*tensor[0];
02580         val -= pre*ux*xr*tensor[1];
02581         val -= pre*uy*yr*tensor[1];
02582         val -= pre*uz*zr*tensor[1];
02583         val += pre*qxx*xr*xr*tensor[2];
02584         val += pre*qyy*yr*yr*tensor[2];
02585         val += pre*qzz*zr*zr*tensor[2];
02586         val += pre*2.0*qxy*xr*yr*tensor[2];
02587         val += pre*2.0*qxz*xr*zr*tensor[2];
02588         val += pre*2.0*qyz*yr*zr*tensor[2];
02589         gzcf[IJKz(i,j,0)] += val;
02590
02591     gpos[2] = zf[nz-1];
02592     zr = gpos[2] - apos[2];
02593     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02594     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02595     val = pre*charge*tensor[0];
02596     val -= pre*ux*xr*tensor[1];
02597     val -= pre*uy*yr*tensor[1];
02598     val -= pre*uz*zr*tensor[1];
02599     val += pre*qxx*xr*xr*tensor[2];
02600     val += pre*qyy*yr*yr*tensor[2];
02601     val += pre*qzz*zr*zr*tensor[2];
02602     val += pre*2.0*qxy*xr*yr*tensor[2];
02603     val += pre*2.0*qxz*xr*zr*tensor[2];
02604     val += pre*2.0*qyz*yr*zr*tensor[2];
02605     gzcf[IJKz(i,j,1)] += val;
02606 }
02607 }
02608 }
```

```

02609
02610 VPRIVATE void bcCalcOrig(Vpmg *thee) {
02611
02612     int nx, ny, nz;
02613     double size, *position, charge, xkappa, eps_w, T, prel;
02614     double *dipole, *quadrupole, debye, eps_p;
02615     double xr,yr,zr,qave,*apos;
02616     double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
02617     int i, j, k, iatom;
02618     Vpbe *pbe;
02619     Vatom *atom;
02620     Valist *alist;
02621
02622     pbe = thee->pbe;
02623     alist = thee->pbe->alist;
02624     nx = thee->pmgp->nx;
02625     ny = thee->pmgp->ny;
02626     nz = thee->pmgp->nz;
02627
02628     /* Zero out the boundaries */
02629     /* the "i" boundaries (dirichlet) */
02630     for (k=0; k<nz; k++) {
02631         for (j=0; j<ny; j++) {
02632             thee->gxcf[IJKx(j,k,0)] = 0.0;
02633             thee->gxcf[IJKx(j,k,1)] = 0.0;
02634             thee->gxcf[IJKx(j,k,2)] = 0.0;
02635             thee->gxcf[IJKx(j,k,3)] = 0.0;
02636         }
02637     }
02638
02639     /* the "j" boundaries (dirichlet) */
02640     for (k=0; k<nz; k++) {
02641         for (i=0; i<nx; i++) {
02642             thee->gycf[IJKy(i,k,0)] = 0.0;
02643             thee->gycf[IJKy(i,k,1)] = 0.0;
02644             thee->gycf[IJKy(i,k,2)] = 0.0;
02645             thee->gycf[IJKy(i,k,3)] = 0.0;
02646         }
02647     }
02648
02649     /* the "k" boundaries (dirichlet) */
02650     for (j=0; j<ny; j++) {
02651         for (i=0; i<nx; i++) {
02652             thee->gzcf[IJKz(i,j,0)] = 0.0;
02653             thee->gzcf[IJKz(i,j,1)] = 0.0;
02654             thee->gzcf[IJKz(i,j,2)] = 0.0;
02655             thee->gzcf[IJKz(i,j,3)] = 0.0;
02656         }
02657     }
02658
02659     /* For each "atom" (only one for bcfl=1), we use the following formula to
02660     * calculate the boundary conditions:
02661     *   g(x) = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_b T} *
02662     *           \exp(-xkappa*(d - a)) / (1 + xkappa*a)
02663     *           * 1/d
02664     * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
02665     * We only need to evaluate some of these prefactors once:

```

```

02666 *      prel = \frac{e_c}{4\pi\epsilon_0\epsilon_w k_b T}
02667 * which gives the potential as
02668 *      g(x) = prel * q/d * \frac{\exp(-xkappa*(d - a))}{1+xkappa*a}
02669 */
02670     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
02671     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
02672     T = Vpbe_getTemperature(pbe);                /* K */
02673     prel = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
02674
02675     /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
02676 * m/A, then we will only need to deal with distances and sizes in
02677 * Angstroms rather than meters. */
02678     xkappa = Vpbe_getXkappa(pbe);                /* A^{-1} */
02679     prel = prel*(1.0e10);
02680
02681     switch (thee->pmgp->bcfl) {
02682         /* If we have zero boundary conditions, we're done */
02683         case BCFL_ZERO:
02684             return;
02685
02686         /* For single DH sphere BC's, we only have one "atom" to deal with;
02687         * get its information and */
02688         case BCFL_SDH:
02689             size = Vpbe_getSoluteRadius(pbe);
02690             position = Vpbe_getSoluteCenter(pbe);
02691
02692             /*
02693             For AMOEBA SDH boundary conditions, we need to find the
02694             total monopole, dipole and traceless quadrupole moments
02695             of either the permanent multipoles, induced dipoles or
02696             non-local induced dipoles.
02697             */
02698
02699             sdhcharge = 0.0;
02700             for (i=0; i<3; i++) sdhdipole[i] = 0.0;
02701             for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
02702
02703             for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
02704                 atom = Valist_getAtom(alist, iatom);
02705                 apos = VatomGetPosition(atom);
02706                 xr = apos[0] - position[0];
02707                 yr = apos[1] - position[1];
02708                 zr = apos[2] - position[2];
02709                 switch (thee->chargeSrc) {
02710                     case VCM_CHARGE:
02711                         charge = Vatom_getCharge(atom);
02712                         sdhcharge += charge;
02713                         sdhdipole[0] += xr * charge;
02714                         sdhdipole[1] += yr * charge;
02715                         sdhdipole[2] += zr * charge;
02716                         traced[0] = xr*xr*charge;
02717                         traced[1] = xr*yr*charge;
02718                         traced[2] = xr*zr*charge;
02719                         traced[3] = yr*xr*charge;
02720                         traced[4] = yr*yr*charge;
02721                         traced[5] = yr*zr*charge;
02722                         traced[6] = zr*xr*charge;

```

```

02723     traced[7] = zr*yr*charge;
02724     traced[8] = zr*zr*charge;
02725     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02726     sdhquadrupole[0] += 1.5*(traced[0] - qave);
02727     sdhquadrupole[1] += 1.5*(traced[1]);
02728     sdhquadrupole[2] += 1.5*(traced[2]);
02729     sdhquadrupole[3] += 1.5*(traced[3]);
02730     sdhquadrupole[4] += 1.5*(traced[4] - qave);
02731     sdhquadrupole[5] += 1.5*(traced[5]);
02732     sdhquadrupole[6] += 1.5*(traced[6]);
02733     sdhquadrupole[7] += 1.5*(traced[7]);
02734     sdhquadrupole[8] += 1.5*(traced[8] - qave);
02735 #if defined(WITH_TINKER)
02736 case VCM_PERMANENT:
02737     charge = Vatom_getCharge(atom);
02738     dipole = Vatom_getDipole(atom);
02739     quadrupole = Vatom_getQuadrupole(atom);
02740     sdhcharge += charge;
02741     sdhdipole[0] += xr * charge;
02742     sdhdipole[1] += yr * charge;
02743     sdhdipole[2] += zr * charge;
02744     traced[0] = xr*xr*charge;
02745     traced[1] = xr*yr*charge;
02746     traced[2] = xr*zr*charge;
02747     traced[3] = yr*xr*charge;
02748     traced[4] = yr*yr*charge;
02749     traced[5] = yr*zr*charge;
02750     traced[6] = zr*xr*charge;
02751     traced[7] = zr*yr*charge;
02752     traced[8] = zr*zr*charge;
02753     sdhdipole[0] += dipole[0];
02754     sdhdipole[1] += dipole[1];
02755     sdhdipole[2] += dipole[2];
02756     traced[0] += 2.0*xr*dipole[0];
02757     traced[1] += xr*dipole[1] + yr*dipole[0];
02758     traced[2] += xr*dipole[2] + zr*dipole[0];
02759     traced[3] += yr*dipole[0] + xr*dipole[1];
02760     traced[4] += 2.0*yr*dipole[1];
02761     traced[5] += yr*dipole[2] + zr*dipole[1];
02762     traced[6] += zr*dipole[0] + xr*dipole[2];
02763     traced[7] += zr*dipole[1] + yr*dipole[2];
02764     traced[8] += 2.0*zr*dipole[2];
02765     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02766     sdhquadrupole[0] += 1.5*(traced[0] - qave);
02767     sdhquadrupole[1] += 1.5*(traced[1]);
02768     sdhquadrupole[2] += 1.5*(traced[2]);
02769     sdhquadrupole[3] += 1.5*(traced[3]);
02770     sdhquadrupole[4] += 1.5*(traced[4] - qave);
02771     sdhquadrupole[5] += 1.5*(traced[5]);
02772     sdhquadrupole[6] += 1.5*(traced[6]);
02773     sdhquadrupole[7] += 1.5*(traced[7]);
02774     sdhquadrupole[8] += 1.5*(traced[8] - qave);
02775     sdhquadrupole[0] += quadrupole[0];
02776     sdhquadrupole[1] += quadrupole[1];
02777     sdhquadrupole[2] += quadrupole[2];
02778     sdhquadrupole[3] += quadrupole[3];
02779     sdhquadrupole[4] += quadrupole[4];

```

```

02780     sdhquadrupole[5] += quadrupole[5];
02781     sdhquadrupole[6] += quadrupole[6];
02782     sdhquadrupole[7] += quadrupole[7];
02783     sdhquadrupole[8] += quadrupole[8];
02784 case VCM_INDUCED:
02785     dipole = Vatom_getInducedDipole(atom);
02786     sdhdipole[0] += dipole[0];
02787     sdhdipole[1] += dipole[1];
02788     sdhdipole[2] += dipole[2];
02789     traced[0] = 2.0*xr*dipole[0];
02790     traced[1] = xr*dipole[1] + yr*dipole[0];
02791     traced[2] = xr*dipole[2] + zr*dipole[0];
02792     traced[3] = yr*dipole[0] + xr*dipole[1];
02793     traced[4] = 2.0*yr*dipole[1];
02794     traced[5] = yr*dipole[2] + zr*dipole[1];
02795     traced[6] = zr*dipole[0] + xr*dipole[2];
02796     traced[7] = zr*dipole[1] + yr*dipole[2];
02797     traced[8] = 2.0*zr*dipole[2];
02798     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02799     sdhquadrupole[0] += 1.5*(traced[0] - qave);
02800     sdhquadrupole[1] += 1.5*(traced[1]);
02801     sdhquadrupole[2] += 1.5*(traced[2]);
02802     sdhquadrupole[3] += 1.5*(traced[3]);
02803     sdhquadrupole[4] += 1.5*(traced[4] - qave);
02804     sdhquadrupole[5] += 1.5*(traced[5]);
02805     sdhquadrupole[6] += 1.5*(traced[6]);
02806     sdhquadrupole[7] += 1.5*(traced[7]);
02807     sdhquadrupole[8] += 1.5*(traced[8] - qave);
02808 case VCM_NLINDUCED:
02809     dipole = Vatom_getNLInducedDipole(atom);
02810     sdhdipole[0] += dipole[0];
02811     sdhdipole[1] += dipole[1];
02812     sdhdipole[2] += dipole[2];
02813     traced[0] = 2.0*xr*dipole[0];
02814     traced[1] = xr*dipole[1] + yr*dipole[0];
02815     traced[2] = xr*dipole[2] + zr*dipole[0];
02816     traced[3] = yr*dipole[0] + xr*dipole[1];
02817     traced[4] = 2.0*yr*dipole[1];
02818     traced[5] = yr*dipole[2] + zr*dipole[1];
02819     traced[6] = zr*dipole[0] + xr*dipole[2];
02820     traced[7] = zr*dipole[1] + yr*dipole[2];
02821     traced[8] = 2.0*zr*dipole[2];
02822     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02823     sdhquadrupole[0] += 1.5*(traced[0] - qave);
02824     sdhquadrupole[1] += 1.5*(traced[1]);
02825     sdhquadrupole[2] += 1.5*(traced[2]);
02826     sdhquadrupole[3] += 1.5*(traced[3]);
02827     sdhquadrupole[4] += 1.5*(traced[4] - qave);
02828     sdhquadrupole[5] += 1.5*(traced[5]);
02829     sdhquadrupole[6] += 1.5*(traced[6]);
02830     sdhquadrupole[7] += 1.5*(traced[7]);
02831     sdhquadrupole[8] += 1.5*(traced[8] - qave);
02832 #endif /* if defined(WITH_TINKER) */
02833 }
02834 }
02835 /* SDH dipole and traceless quadrupole values
02836 were checked against similar routines in TINKER

```

```

02837     for large proteins.
02838
02839     debbye=4.8033324;
02840     printf("%6.3f, %6.3f, %6.3f\n", sdhdipole[0]*debbye,
02841         sdhdipole[1]*debbye, sdhdipole[2]*debbye);
02842     printf("%6.3f\n", sdhquadrupole[0]*debbye);
02843     printf("%6.3f %6.3f\n", sdhquadrupole[3]*debbye,
02844         sdhquadrupole[4]*debbye);
02845     printf("%6.3f %6.3f %6.3f\n", sdhquadrupole[6]*debbye,
02846         sdhquadrupole[7]*debbye, sdhquadrupole[8]*debbye);
02847     */
02848
02849     bcfl2(size, position, sdhcharge, sdhdipole, sdhquadrupole,
02850         xkappa, eps_p, eps_w, T, thee->gxcf, thee->gycf,
02851         thee->gzcf, thee->xf, thee->yf, thee->zf, nx, ny, nz);
02852     break;
02853
02854     case BCFL_MDH:
02855     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
02856         atom = Valist_getAtom(alist, iatom);
02857         position = Vatom_getPosition(atom);
02858         charge = Vunit_ec*Vatom_getCharge(atom);
02859         dipole = VNULL;
02860         quadrupole = VNULL;
02861         size = Vatom_getRadius(atom);
02862         switch (thee->chargeSrc)
02863     {
02864         case VCM_CHARGE:
02865         ;
02866 #if defined(WITH_TINKER)
02867         case VCM_PERMANENT:
02868             dipole = Vatom_getDipole(atom);
02869             quadrupole = Vatom_getQuadrupole(atom);
02870
02871         case VCM_INDUCED:
02872             dipole = Vatom_getInducedDipole(atom);
02873
02874         case VCM_NLINDUCED:
02875             dipole = Vatom_getNLInducedDipole(atom);
02876 #endif
02877     }
02878     bcfl1(size, position, charge, xkappa, prel,
02879         thee->gxcf, thee->gycf, thee->gzcf,
02880         thee->xf, thee->yf, thee->zf, nx, ny, nz);
02881 }
02882 break;
02883
02884     case BCFL_UNUSED:
02885         Vnm_print(2, "bcCalc: Invalid bcfl (%d)!\n", thee->pmgp->bcfl);
02886         VASSERT(0);
02887
02888     case BCFL_FOCUS:
02889         Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
02890         VASSERT(0);
02891
02892     default:
02893         Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \

```

```
02894 flag (%d) !\n", thee->pmgp->bcfl);
02895             VASSERT(0);
02896         }
02897     }
02898
02899 /* 
02900 Used by bcflnew
02901 */
02902 VPRIvATE int gridPointIsValid(int i, int j, int k, int nx, int ny, int nz){
02903
02904     int isValid = 0;
02905
02906     if((k==0) || (k==nz-1)){
02907         isValid = 1;
02908     }else if((j==0) || (j==ny-1)){
02909         isValid = 1;
02910     }else if((i==0) || (i==nx-1)){
02911         isValid = 1;
02912     }
02913
02914     return isValid;
02915 }
02916
02917 /*
02918 Used by bcflnew
02919 */
02920 #ifdef DEBUG_MAC OSX_OCL
02921 #include "mach_chud.h"
02922 VPRIvATE void packAtomsOpenCL(float *ax, float *ay, float *az,
02923         float *charge, float *size, Vpmg *thee){
02924
02925     int i;
02926     int natoms;
02927
02928     Vatom *atom = VNULL;
02929     Valist *alist = VNULL;
02930
02931     alist = thee->pbe->alist;
02932     natoms = Valist_getNumberAtoms(alist);
02933
02934     for(i=0;i<natoms;i++){
02935         atom = &(alist->atoms[i]);
02936         charge[i] = Vunit_ec*atom->charge;
02937         ax[i] = atom->position[0];
02938         ay[i] = atom->position[1];
02939         az[i] = atom->position[2];
02940         size[i] = atom->radius;
02941     }
02942 }
02943
02944 /*
02945 Used by bcflnew
02946 */
02947 VPRIvATE void packUnpackOpenCL(int nx, int ny, int nz, int ngrid,
02948         float *gx, float *gy, float *gz, float *value,
02949         Vpmg *thee, int pack){
02950 }
```

```

02951 int i,j,k,igrid;
02952 int x0,x1,y0,y1,z0,z1;
02953
02954 float gpos[3];
02955 double *xf, *yf, *zf;
02956 double *gxcf, *gycf, *gzcf;
02957
02958 xf = thee->xf;
02959 yf = thee->yf;
02960 zf = thee->zf;
02961
02962 gxcf = thee->gxcf;
02963 gycf = thee->gycf;
02964 gzcf = thee->gzcf;
02965
02966 igrid = 0;
02967 for(k=0;k<nz;k++) {
02968   gpos[2] = zf[k];
02969   for(j=0;j<ny;j++) {
02970     gpos[1] = yf[j];
02971     for(i=0;i<nx;i++) {
02972       gpos[0] = xf[i];
02973       if(gridPointIsValid(i, j, k, nx, ny, nz)) {
02974         if(pack != 0) {
02975           gx[igrid] = gpos[0];
02976           gy[igrid] = gpos[1];
02977           gz[igrid] = gpos[2];
02978
02979           value[igrid] = 0.0;
02980         }else{
02981           x0 = IJKx(j,k,0);
02982           x1 = IJKx(j,k,1);
02983           y0 = IJKy(i,k,0);
02984           y1 = IJKy(i,k,1);
02985           z0 = IJKz(i,j,0);
02986           z1 = IJKz(i,j,1);
02987
02988           if(i==0) {
02989             gxcf[x0] += value[igrid];
02990           }
02991           if(i==nx-1) {
02992             gxcf[x1] += value[igrid];
02993           }
02994           if(j==0) {
02995             gycf[y0] += value[igrid];
02996           }
02997           if(j==ny-1) {
02998             gycf[y1] += value[igrid];
02999           }
03000           if(k==0) {
03001             gzcf[z0] += value[igrid];
03002           }
03003           if(k==nz-1) {
03004             gzcf[z1] += value[igrid];
03005           }
03006         }
03007       }

```

```
03008     igridd++;
03009 } //end is valid point
03010 } //end i
03011 } //end j
03012 } //end k
03013
03014 }
03015
03016 /* 
03017 bcflnew is an optimized replacement for bcfl1. bcfl1 is still used when TINKER
03018 support is compiled in.
03019 bcflnew uses: packUnpack, packAtoms, gridPointIsValid
03020 */
03021 VPRIVATE void bcflnewOpenCL(Vpmg *thee) {
03022
03023 int i,j,k, iatom, igridd;
03024 int x0, x1, y0, y1, z0, z1;
03025
03026 int nx, ny, nz;
03027 int natoms, ngrid, ngadj;
03028
03029 float dist, prel, eps_w, eps_p, T, xkappa;
03030
03031 float *ax, *ay, *az;
03032 float *charge, *size, *val;
03033
03034 float *gx, *gy, *gz;
03035
03036 Vpbe *pbe = thee->pbe;
03037
03038 nx = thee->pmgp->nx;
03039 ny = thee->pmgp->ny;
03040 nz = thee->pmgp->nz;
03041
03042 eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03043 eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03044 T = Vpbe_getTemperature(pbe);                /* K */
03045 prel = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
03046 xkappa = Vpbe_getXkappa(pbe);
03047
03048 natoms = Valist_getNumberAtoms(thee->pbe->alist);
03049 ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
03050 ngadj = ngrid + (512 - (ngrid & 511));
03051
03052 ax = (float*)malloc(natoms * sizeof(float));
03053 ay = (float*)malloc(natoms * sizeof(float));
03054 az = (float*)malloc(natoms * sizeof(float));
03055
03056 charge = (float*)malloc(natoms * sizeof(float));
03057 size = (float*)malloc(natoms * sizeof(float));
03058
03059 gx = (float*)malloc(ngrid * sizeof(float));
03060 gy = (float*)malloc(ngrid * sizeof(float));
03061 gz = (float*)malloc(ngrid * sizeof(float));
03062
03063 val = (float*)malloc(ngrid * sizeof(float));
03064
```

```

03065 packAtomsOpenCL(ax,ay,az,charge,size,thee);
03066 packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);
03067
03068 runMDHCL(ngrid,natoms,ngadj,ax,ay,az,gx,gy,gz,charge,size,xkappa,prel,val);
03069
03070 packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);
03071
03072 free(ax);
03073 free(ay);
03074 free(az);
03075 free(charge);
03076 free(size);
03077
03078 free(gx);
03079 free(gy);
03080 free(gz);
03081 free(val);
03082 }
03083 #endif
03084
03085 VPRIVATE void packAtoms(double *ax, double *ay, double *az,
03086     double *charge, double *size, Vpmg *thee){
03087
03088 int i;
03089 int natoms;
03090
03091 Vatom *atom = VNULL;
03092 Valist *alist = VNULL;
03093
03094 alist = thee->pbe->alist;
03095 natoms = Valist_getNumberAtoms(alist);
03096
03097 for(i=0;i<natoms;i++){
03098 atom = &(alist->atoms[i]);
03099 charge[i] = Vunit_ec*atom->charge;
03100 ax[i] = atom->position[0];
03101 ay[i] = atom->position[1];
03102 az[i] = atom->position[2];
03103 size[i] = atom->radius;
03104 }
03105 }
03106
03107 /*
03108 Used by bcflnew
03109 */
03110 VPRIVATE void packUnpack(int nx, int ny, int nz, int ngrid,
03111     double *gx, double *gy, double *gz, double *value,
03112     Vpmg *thee, int pack){
03113
03114 int i,j,k,igrid;
03115 int x0,x1,y0,y1,z0,z1;
03116
03117 double gpos[3];
03118 double *xf, *yf, *zf;
03119 double *gxcf, *gycf, *gzcf;
03120
03121 xf = thee->xf;

```

```
03122     yf = thee->yf;
03123     zf = thee->zf;
03124
03125     gxcf = thee->gxcf;
03126     gycf = thee->gycf;
03127     gzcf = thee->gzcf;
03128
03129     igrd = 0;
03130     for(k=0;k<nz;k++) {
03131         gpos[2] = zf[k];
03132         for(j=0;j<ny;j++) {
03133             gpos[1] = yf[j];
03134             for(i=0;i<nx;i++) {
03135                 gpos[0] = xf[i];
03136                 if(gridPointIsValid(i, j, k, nx, ny, nz)) {
03137                     if(pack != 0) {
03138                         gx[igrd] = gpos[0];
03139                         gy[igrd] = gpos[1];
03140                         gz[igrd] = gpos[2];
03141
03142                         value[igrd] = 0.0;
03143                     } else{
03144                         x0 = IJKx(j,k,0);
03145                         x1 = IJKx(j,k,1);
03146                         y0 = IJKy(i,k,0);
03147                         y1 = IJKy(i,k,1);
03148                         z0 = IJKz(i,j,0);
03149                         z1 = IJKz(i,j,1);
03150
03151                         if(i==0) {
03152                             gxcf[x0] += value[igrd];
03153                         }
03154                         if(i==nx-1) {
03155                             gxcf[x1] += value[igrd];
03156                         }
03157                         if(j==0) {
03158                             gycf[y0] += value[igrd];
03159                         }
03160                         if(j==ny-1) {
03161                             gycf[y1] += value[igrd];
03162                         }
03163                         if(k==0) {
03164                             gzcf[z0] += value[igrd];
03165                         }
03166                         if(k==nz-1) {
03167                             gzcf[z1] += value[igrd];
03168                         }
03169                     }
03170
03171                     igrd++;
03172                 } //end is valid point
03173             } //end i
03174         } //end j
03175     } //end k
03176
03177 }
03178
```

```

03179 VPRIVATE void bcflnew(Vpmg *thee) {
03180
03181     int i,j,k, iatom, igrd;
03182     int x0, x1, y0, y1, z0, z1;
03183
03184     int nx, ny, nz;
03185     int natoms, ngrid;
03186
03187     double dist, prel, eps_w, eps_p, T, xkappa;
03188
03189     double *ax, *ay, *az;
03190     double *charge, *size, *val;
03191
03192     double *gx, *gy, *gz;
03193
03194     Vpbe *pbe = thee->pbe;
03195
03196     nx = thee->pmgp->nx;
03197     ny = thee->pmgp->ny;
03198     nz = thee->pmgp->nz;
03199
03200     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03201     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03202     T = Vpbe_getTemperature(pbe);                /* K */
03203     prel = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
03204     xkappa = Vpbe_getXkappa(pbe);
03205
03206     natoms = Valist_getNumberAtoms(thee->pbe->alist);
03207     ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
03208
03209     ax = (double*)malloc(natoms * sizeof(double));
03210     ay = (double*)malloc(natoms * sizeof(double));
03211     az = (double*)malloc(natoms * sizeof(double));
03212
03213     charge = (double*)malloc(natoms * sizeof(double));
03214     size = (double*)malloc(natoms * sizeof(double));
03215
03216     gx = (double*)malloc(ngrid * sizeof(double));
03217     gy = (double*)malloc(ngrid * sizeof(double));
03218     gz = (double*)malloc(ngrid * sizeof(double));
03219
03220     val = (double*)malloc(ngrid * sizeof(double));
03221
03222     packAtoms(ax,ay,az,charge,size,thee);
03223     packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);
03224
03225     if(xkappa > VSMALL){
03226 #pragma omp parallel for default(shared) private(igrd,iatom,dist)
03227         for(igrd=0;igrd<ngrid;igrd++){
03228             for(iatom=0; iatom<natoms; iatom++){
03229                 dist = VSQRT(VSQR(gx[igrd]-ax[iatom]) + VSQR(gy[igrd]-ay[iatom])
03230                         + VSQR(gz[igrd]-az[iatom]));
03231                 val[igrd] += prel*(charge[iatom]/dist)*VEXP(-xkappa*(dist-size[iatom]))
03232                     / (1+xkappa*size[iatom]);
03233             }
03234         }
03235     }else{

```

```

03236 #pragma omp parallel for default(shared) private(igrid,iatom,dist)
03237     for(igrid=0;igrid<ngrid;igrid++) {
03238         for(iatom=0; iatom<natoms; iatom++) {
03239             dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
03240                 + VSQR(gz[igrid]-az[iatom]));
03241             val[igrid] += pre1*(charge[iatom]/dist);
03242         }
03243     }
03244 }
03245 packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);
03246
03247 free(ax);
03248 free/ay);
03249 free(az);
03250 free(charge);
03251 free(size);
03252
03253 free(gx);
03254 free(gy);
03255 free(gz);
03256 free(val);
03257 }
03258
03259 VPRIATE void multipolebc(double r, double kappa, double eps_p,
03260                             double eps_w, double rad, double tsr[3]) {
03261     double r2,r3,r5;
03262     double eps_r;
03263     double ka,ka2,ka3;
03264     double kr,kr2,kr3;
03265
03266     /*
03267     Below an attempt is made to explain the potential outside of a
03268     multipole located at the center of spherical cavity of dielectric
03269     eps_p, with dielectric eps_w outside (and possibly kappa > 0).
03270
03271
03272 First, eps_p = 1.0
03273 eps_w = 1.0
03274 kappa = 0.0
03275
03276 The general form for the potential of a traceless multipole tensor
03277 of rank n in vacuum is:
03278
03279 V(r) = (-1)^n * u . n . Del^n (1/r)
03280
03281 where
03282 u           is a multipole of order n (3^n components)
03283 u . n . Del^n (1/r)   is the contraction of u with the nth
03284 derivative of 1/r
03285
03286 for example, if n = 1, the dipole potential is
03287 V_vac(r) = (-1)*[ux*x + uy*y + uz*z]/r^3
03288
03289 This function returns the parts of V(r) for multipoles of
03290 order 0, 1 and 2 that are independent of the contraction.
03291
03292 For the vacuum example, this would be 1/r, -1/r^3 and 3/r^5

```

```

03293    respectively.
03294
03295    *** Note that this requires that the quadrupole is
03296    traceless. If not, the diagonal quadrupole potential changes
03297    from
03298    qaa * 3*a^2/r^5
03299    to
03300    qaa * (3*a^2/r^5 - 1/r^3a )
03301    where we sum over the trace; a = x, y and z.
03302
03303    (In other words, the -1/r^3 term cancels for a traceless quadrupole.
03304    qxx + qyy + qzz = 0
03305    such that
03306    -(qxx + qyy + qzz)/r^3 = 0
03307
03308    For quadrupole with trace:
03309    qxx + qyy + qzz != 0
03310    such that
03311    -(qxx + qyy + qzz)/r^3 != 0
03312    )
03313
03314 =====
03315
03316    eps_p != 1 or eps_w != 1
03317    kappa = 0.0
03318
03319    If the multipole is placed at the center of a sphere with
03320    dielectric eps_p in a solvent of dielectric eps_w, the potential
03321    outside the sphere is the solution to the Laplace equation:
03322
03323    V(r) = 1/eps_w * (2*n+1)*eps_r/(n+(n+1)*eps_r)
03324    * (-1)^n * u . n . Del^n (1/r)
03325    where
03326    eps_r = eps_w / eps_p
03327    is the ratio of solvent to solute dielectric
03328
03329 =====
03330
03331    kappa > 0
03332
03333    Finally, if the region outside the sphere is treated by the linearized
03334    PB equation with Debye-Huckel parameter kappa, the solution is:
03335
03336    V(r) = kappa/eps_w * alpha_n(kappa*a) * K_n(kappa*r) * r^(n+1)/a^n
03337    * (-1)^n * u . n . Del^n (1/r)
03338    where
03339    alpha_n(x) is [(2n + 1) / x] / [(n*K_n(x)/eps_r) - x*K_n'(x)]
03340    K_n(x) are modified spherical Bessel functions of the third kind.
03341    K_n'(x) is the derivative of K_n(x)
03342    */
03343
03344    eps_r = eps_w/eps_p;
03345    r2 = r*r;
03346    r3 = r2*r;
03347    r5 = r3*r2;
03348    tsr[0] = (1.0/eps_w)/r;
03349    tsr[1] = (1.0/eps_w)*(-1.0)/r3;

```

```

03350     tsr[2] = (1.0/eps_w)*(3.0)/r5;
03351     if (kappa < VSMALL) {
03352         tsr[1] = (3.0*eps_r)/(1.0 + 2.0*eps_r)*tsr[1];
03353         tsr[2] = (5.0*eps_r)/(2.0 + 3.0*eps_r)*tsr[2];
03354     } else {
03355         ka = kappa*rad;
03356         ka2 = ka*ka;
03357         ka3 = ka2*ka;
03358         kr = kappa*r;
03359         kr2 = kr*kr;
03360         kr3 = kr2*kr;
03361         tsr[0] = exp(ka-kr) / (1.0 + ka) * tsr[0];
03362         tsr[1] = 3.0*eps_r*exp(ka-kr)*(1.0 + kr) * tsr[1];
03363         tsr[1] = tsr[1] / (1.0 + ka + eps_r*(2.0 + 2.0*ka + ka2));
03364         tsr[2] = 5.0*eps_r*exp(ka-kr)*(3.0 + 3.0*kr + kr2) * tsr[2];
03365         tsr[2] = tsr[2]/(6.0+6.0*ka+2.0*ka2+eps_r*(9.0+9.0*ka+4.0*ka2+ka3));
03366     }
03367 }
03368
03369 VPRIVATE void bcfl_sdh(Vpmg *thee) {
03370
03371     int i,j,k,iatom;
03372     int nx, ny, nz;
03373
03374     double size, *position, charge, xkappa, eps_w, eps_p, T, pre, dist;
03375     double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
03376     double *dipole, *quadrupole;
03377
03378     double val, *apos, gpos[3], tensor[3], qave;
03379     double ux, uy, uz, xr, yr, zr;
03380     double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
03381
03382     double *xf, *yf, *zf;
03383     double *gxcf, *gycf, *gzcf;
03384
03385     Vpbe *pbe;
03386     Vatom *atom;
03387     Valist *alist;
03388
03389     pbe = thee->pbe;
03390     alist = thee->pbe->alist;
03391     nx = thee->pmgp->nx;
03392     ny = thee->pmgp->ny;
03393     nz = thee->pmgp->nz;
03394
03395     xf = thee->xf;
03396     yf = thee->yf;
03397     zf = thee->zf;
03398
03399     gxcf = thee->gxcf;
03400     gycf = thee->gycf;
03401     gzcf = thee->gzcf;
03402
03403     /* For each "atom" (only one for bcfl=1), we use the following formula to
03404      * calculate the boundary conditions:
03405      *      g(x) = \frac{q e_c}{4\pi \epsilon_0 \epsilon_w k_b T} *
03406      *              \frac{\exp(-xkappa*(d - a))}{1+xkappa*a}

```

```

03407      *           * 1/d
03408      * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
03409      * We only need to evaluate some of these prefactors once:
03410      *     pre1 = \frac{e_c}{4\pi\epsilon_0\epsilon_w k_b T}
03411      * which gives the potential as
03412      *     g(x) = pre1 * q/d * \frac{\exp(-xkappa*(d - a))}{(1+xkappa*a)}
03413      */
03414      eps_w = Vpbe_getSolventDiel(pbe);          /* Dimensionless */
03415      eps_p = Vpbe_getSoluteDiel(pbe);          /* Dimensionless */
03416      T = Vpbe_getTemperature(pbe);              /* K */
03417
03418      pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
03419      pre = pre*(1.0e10);
03420
03421      /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
03422      * m/A, then we will only need to deal with distances and sizes in
03423      * Angstroms rather than meters. */
03424      xkappa = Vpbe_getXkappa(pbe);             /* A^{-1} */
03425
03426      /* Solute size and position */
03427      size = Vpbe_getSoluteRadius(pbe);
03428      position = Vpbe_getSoluteCenter(pbe);
03429
03430      sdhcharge = 0.0;
03431      for (i=0; i<3; i++) sdhdipole[i] = 0.0;
03432      for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
03433
03434      for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03435          atom = Valist_getAtom(alist, iatom);
03436          apos = Vatom_getPosition(atom);
03437          xr = apos[0] - position[0];
03438          yr = apos[1] - position[1];
03439          zr = apos[2] - position[2];
03440          switch (thee->chargeSrc) {
03441              case VCM_CHARGE:
03442                  charge = Vatom_getCharge(atom);
03443                  sdhcharge += charge;
03444                  sdhdipole[0] += xr * charge;
03445                  sdhdipole[1] += yr * charge;
03446                  sdhdipole[2] += zr * charge;
03447                  traced[0] = xr*xr*charge;
03448                  traced[1] = xr*yr*charge;
03449                  traced[2] = xr*zr*charge;
03450                  traced[3] = yr*xr*charge;
03451                  traced[4] = yr*yr*charge;
03452                  traced[5] = yr*zr*charge;
03453                  traced[6] = zr*xr*charge;
03454                  traced[7] = zr*yr*charge;
03455                  traced[8] = zr*zr*charge;
03456                  qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03457                  sdhquadrupole[0] += 1.5*(traced[0] - qave);
03458                  sdhquadrupole[1] += 1.5*(traced[1]);
03459                  sdhquadrupole[2] += 1.5*(traced[2]);
03460                  sdhquadrupole[3] += 1.5*(traced[3]);
03461                  sdhquadrupole[4] += 1.5*(traced[4] - qave);
03462                  sdhquadrupole[5] += 1.5*(traced[5]);
03463                  sdhquadrupole[6] += 1.5*(traced[6]);

```

```

03464     sdhquadrupole[7] += 1.5*(traced[7]);
03465     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03466 #if defined(WITH_TINKER)
03467     case VCM_PERMANENT:
03468         charge = Vatom_getCharge(atom);
03469         dipole = Vatom_getDipole(atom);
03470         quadrupole = Vatom_getQuadrupole(atom);
03471         sdhcharge += charge;
03472         sdhdipole[0] += xr * charge;
03473         sdhdipole[1] += yr * charge;
03474         sdhdipole[2] += zr * charge;
03475         traced[0] = xr*xr*charge;
03476         traced[1] = xr*yr*charge;
03477         traced[2] = xr*zr*charge;
03478         traced[3] = yr*xr*charge;
03479         traced[4] = yr*yr*charge;
03480         traced[5] = yr*zr*charge;
03481         traced[6] = zr*xr*charge;
03482         traced[7] = zr*yr*charge;
03483         traced[8] = zr*zr*charge;
03484         sdhdipole[0] += dipole[0];
03485         sdhdipole[1] += dipole[1];
03486         sdhdipole[2] += dipole[2];
03487         traced[0] += 2.0*xr*dipole[0];
03488         traced[1] += xr*dipole[1] + yr*dipole[0];
03489         traced[2] += xr*dipole[2] + zr*dipole[0];
03490         traced[3] += yr*dipole[0] + xr*dipole[1];
03491         traced[4] += 2.0*yr*dipole[1];
03492         traced[5] += yr*dipole[2] + zr*dipole[1];
03493         traced[6] += zr*dipole[0] + xr*dipole[2];
03494         traced[7] += zr*dipole[1] + yr*dipole[2];
03495         traced[8] += 2.0*zr*dipole[2];
03496         qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03497         sdhquadrupole[0] += 1.5*(traced[0] - qave);
03498         sdhquadrupole[1] += 1.5*(traced[1]);
03499         sdhquadrupole[2] += 1.5*(traced[2]);
03500         sdhquadrupole[3] += 1.5*(traced[3]);
03501         sdhquadrupole[4] += 1.5*(traced[4] - qave);
03502         sdhquadrupole[5] += 1.5*(traced[5]);
03503         sdhquadrupole[6] += 1.5*(traced[6]);
03504         sdhquadrupole[7] += 1.5*(traced[7]);
03505         sdhquadrupole[8] += 1.5*(traced[8] - qave);
03506         sdhquadrupole[0] += quadrupole[0];
03507         sdhquadrupole[1] += quadrupole[1];
03508         sdhquadrupole[2] += quadrupole[2];
03509         sdhquadrupole[3] += quadrupole[3];
03510         sdhquadrupole[4] += quadrupole[4];
03511         sdhquadrupole[5] += quadrupole[5];
03512         sdhquadrupole[6] += quadrupole[6];
03513         sdhquadrupole[7] += quadrupole[7];
03514         sdhquadrupole[8] += quadrupole[8];
03515     case VCM_INDUCED:
03516         dipole = Vatom_getInducedDipole(atom);
03517         sdhdipole[0] += dipole[0];
03518         sdhdipole[1] += dipole[1];
03519         sdhdipole[2] += dipole[2];
03520         traced[0] = 2.0*xr*dipole[0];

```

```

03521     traced[1] = xr*dipole[1] + yr*dipole[0];
03522     traced[2] = xr*dipole[2] + zr*dipole[0];
03523     traced[3] = yr*dipole[0] + xr*dipole[1];
03524     traced[4] = 2.0*yr*dipole[1];
03525     traced[5] = yr*dipole[2] + zr*dipole[1];
03526     traced[6] = zr*dipole[0] + xr*dipole[2];
03527     traced[7] = zr*dipole[1] + yr*dipole[2];
03528     traced[8] = 2.0*zr*dipole[2];
03529     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03530     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03531     sdhquadrupole[1] += 1.5*(traced[1]);
03532     sdhquadrupole[2] += 1.5*(traced[2]);
03533     sdhquadrupole[3] += 1.5*(traced[3]);
03534     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03535     sdhquadrupole[5] += 1.5*(traced[5]);
03536     sdhquadrupole[6] += 1.5*(traced[6]);
03537     sdhquadrupole[7] += 1.5*(traced[7]);
03538     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03539 case VCM_NLINDUCED:
03540     dipole = Vatom_getNLInducedDipole(atom);
03541     sdhdipole[0] += dipole[0];
03542     sdhdipole[1] += dipole[1];
03543     sdhdipole[2] += dipole[2];
03544     traced[0] = 2.0*xr*dipole[0];
03545     traced[1] = xr*dipole[1] + yr*dipole[0];
03546     traced[2] = xr*dipole[2] + zr*dipole[0];
03547     traced[3] = yr*dipole[0] + xr*dipole[1];
03548     traced[4] = 2.0*yr*dipole[1];
03549     traced[5] = yr*dipole[2] + zr*dipole[1];
03550     traced[6] = zr*dipole[0] + xr*dipole[2];
03551     traced[7] = zr*dipole[1] + yr*dipole[2];
03552     traced[8] = 2.0*zr*dipole[2];
03553     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03554     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03555     sdhquadrupole[1] += 1.5*(traced[1]);
03556     sdhquadrupole[2] += 1.5*(traced[2]);
03557     sdhquadrupole[3] += 1.5*(traced[3]);
03558     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03559     sdhquadrupole[5] += 1.5*(traced[5]);
03560     sdhquadrupole[6] += 1.5*(traced[6]);
03561     sdhquadrupole[7] += 1.5*(traced[7]);
03562     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03563 #endif /* if defined(WITH_TINKER) */
03564 }
03565 }
03566
03567 ux = sdhdipole[0];
03568 uy = sdhdipole[1];
03569 uz = sdhdipole[2];
03570
03571 /* The factor of 1/3 results from using a
03572    traceless quadrupole definition. See, for example,
03573    "The Theory of Intermolecular Forces" by A.J. Stone,
03574    Chapter 3. */
03575 qxx = sdhquadrupole[0] / 3.0;
03576 qxy = sdhquadrupole[1] / 3.0;
03577 qxz = sdhquadrupole[2] / 3.0;

```

```

03578     qyx = sdhquadrupole[3] / 3.0;
03579     qyy = sdhquadrupole[4] / 3.0;
03580     qyz = sdhquadrupole[5] / 3.0;
03581     qzx = sdhquadrupole[6] / 3.0;
03582     qzy = sdhquadrupole[7] / 3.0;
03583     qzz = sdhquadrupole[8] / 3.0;
03584
03585     for (k=0;k<nz;k++) {
03586         gpos[2] = zf[k];
03587         for (j=0;j<ny;j++) {
03588             gpos[1] = yf[j];
03589             for (i=0;i<nx;i++) {
03590                 gpos[0] = xf[i];
03591                 if (gridPointIsValid(i, j, k, nx, ny, nz)) {
03592                     xr = gpos[0] - position[0];
03593                     yr = gpos[1] - position[1];
03594                     zr = gpos[2] - position[2];
03595
03596                     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
03597                     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
03598
03599                     val = pre*sdhcharge*tensor[0];
03600                     val -= pre*sux*xr*tensor[1];
03601                     val -= pre*uy*yr*tensor[1];
03602                     val -= pre*uz*zr*tensor[1];
03603                     val += pre*qxx*xr*xr*tensor[2];
03604                     val += pre*qyy*yr*yr*tensor[2];
03605                     val += pre*qzz*zr*zr*tensor[2];
03606                     val += pre*2.0*qxy*xr*yr*tensor[2];
03607                     val += pre*2.0*qxz*xr*zr*tensor[2];
03608                     val += pre*2.0*qyz*yr*zr*tensor[2];
03609
03610                     if (i==0) {
03611                         gxcf[IJKx(j,k,0)] = val;
03612                     }
03613                     if (i==nx-1) {
03614                         gxcf[IJKx(j,k,1)] = val;
03615                     }
03616                     if (j==0) {
03617                         gycf[IJKy(i,k,0)] = val;
03618                     }
03619                     if (j==ny-1) {
03620                         gycf[IJKy(i,k,1)] = val;
03621                     }
03622                     if (k==0) {
03623                         gzcf[IJKz(i,j,0)] = val;
03624                     }
03625                     if (k==nz-1) {
03626                         gzcf[IJKz(i,j,1)] = val;
03627                     }
03628                     /* End grid point is valid */
03629                 } /* End i loop */
03630             } /* End j loop */
03631         } /* End k loop */
03632
03633     }
03634

```

```

03635 VPRIVATE void bcfl_mdh(Vpmg *thee) {
03636     int i,j,k,iatom;
03637     int nx, ny, nz;
03638
03639     double val, *apos, gpos[3];
03640     double *dipole, *quadrupole;
03641     double size, charge, xkappa, eps_w, eps_p, T, prel, dist;
03642
03643     double *xf, *yf, *zf;
03644     double *gxcf, *gycf, *gzcf;
03645
03646     Vpbe *pbe;
03647     Vatom *atom;
03648     Valist *alist;
03649
03650     pbe = thee->pbe;
03651     alist = thee->pbe->alist;
03652     nx = thee->pmgp->nx;
03653     ny = thee->pmgp->ny;
03654     nz = thee->pmgp->nz;
03655
03656     xf = thee->xf;
03657     yf = thee->yf;
03658     zf = thee->zf;
03659
03660     gxcf = thee->gxcf;
03661     gycf = thee->gycf;
03662     gzcf = thee->gzcf;
03663
03664     /* For each "atom" (only one for bcfl=1), we use the following formula to
03665      * calculate the boundary conditions:
03666      *   g(x) = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_b T} \exp(-xkappa*(d - a)) / (1 + xkappa*a)
03667      *   where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
03668      *   We only need to evaluate some of these prefactors once:
03669      *   prel = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_b T}
03670      *   which gives the potential as
03671      *   g(x) = prel * q/d * \frac{\exp(-xkappa*(d - a))}{(1 + xkappa*a)}
03672      */
03673
03674     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03675     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03676     T = Vpbe_getTemperature(pbe);                /* K */
03677     prel = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
03678
03679     /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
03680      * m/A, then we will only need to deal with distances and sizes in
03681      * Angstroms rather than meters. */
03682     xkappa = Vpbe_getXkappa(pbe);               /* A^{-1} */
03683     prel = prel*(1.0e10);
03684
03685     /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
03686      * m/A, then we will only need to deal with distances and sizes in
03687      * Angstroms rather than meters. */
03688     xkappa = Vpbe_getXkappa(pbe);               /* A^{-1} */
03689
03690
03691

```

```

03692   for(k=0;k<nz;k++) {
03693     gpos[2] = zf[k];
03694     for(j=0;j<ny;j++) {
03695       gpos[1] = yf[j];
03696       for(i=0;i<nx;i++) {
03697         gpos[0] = xf[i];
03698         if(gridPointIsValid(i, j, k, nx, ny, nz)) {
03699           val = 0.0;
03700           for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03701             atom = Valist_getAtom(alist, iatom);
03702             apos = Vatom_getPosition(atom);
03703             charge = Vunit_ec*Vatom_getCharge(atom);
03704             size = Vatom_getRadius(atom);
03705             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
03706                         + VSQR(gpos[2]-apos[2]));
03707             if (xkappa > VSMALL) {
03708               val += pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
03709                         / (1+xkappa*size);
03710             } else {
03711               val += pre1*(charge/dist);
03712             }
03713           }
03714           if(i==0){
03715             gxcf[IJKx(j,k,0)] = val;
03716           }
03717           if(i==nx-1){
03718             gxcf[IJKx(j,k,1)] = val;
03719           }
03720           if(j==0){
03721             gycf[IJKy(i,k,0)] = val;
03722           }
03723           if(j==ny-1){
03724             gycf[IJKy(i,k,1)] = val;
03725           }
03726           if(k==0){
03727             gzcf[IJKz(i,j,0)] = val;
03728           }
03729           if(k==nz-1){
03730             gzcf[IJKz(i,j,1)] = val;
03731           }
03732           /* End grid point is valid */
03733         } /* End i loop */
03734       } /* End j loop */
03735     } /* End k loop */
03736   }
03737   /* ////////////////////////////// */
03738   // Routine: bcfl_mem
03739   //
03740   // Purpose: Increment all the boundary points by the
03741   // analytic expression for a membrane system in

```

```

03749 // the presence of a membrane potential. This
03750 // Boundary flag should only be used for systems
03751 // that explicitly have membranes in the dielectric
03752 // and solvent maps.
03753 //
03754 // There should be several input variables add to this
03755 // function such as membrane potential, membrane thickness
03756 // and height.
03757 //
03758 // Args:      apos is a 3-vector
03759 //
03760 // Author: Michael Grabe
03762 VPRIIVATE void bcfl_mem(double zmem, double L, double eps_m, double eps_w,
03763     double V, double xkappa, double *gxcf, double *gycf, double *gzcf,
03764     double *xf, double *yf, double *zf, int nx, int ny, int nz) {
03765
03767 /* some definitions */ */
03768 /* L = total length of the membrane */ */
03769 /* xkappa = inverse Debeye length */ */
03770 /* zmem = z value of membrane bottom (Cytoplasm) */ */
03771 /* V = electrical potential inside the cell */ */
03773 int i, j, k;
03774 double dist, val, z_low, z_high, z_shift;
03775 double A, B, C, D, edge_L, l;
03776 double G, z_0, z_rel;
03777 double gpos[3];
03778
03779 printf("Here is the value of kappa: %f\n",xkappa);
03780 printf("Here is the value of L: %f\n",L);
03781 printf("Here is the value of zmem: %f\n",zmem);
03782 printf("Here is the value of mdie: %f\n",eps_m);
03783 printf("Here is the value of memv: %f\n",V);
03784
03785 /* no salt symmetric BC's at +/- infinity */
03786 // B=V/(edge_L - l*(1-eps_w/eps_m));
03787 // A=V + B*edge_L;
03788 // D=eps_w/eps_m*B;
03789 z_low = zmem; /* this defines the bottom of the membrane */
03790 z_high = zmem + L; /* this is the top of the membrane */
03791
03792 /***** */
03793 /* proper boundary conditions for V = 0 extracellular */
03794 /* and psi=-V cytoplasm. */
03795 /* Implicit in this formulation is that the membrane */
03796 /* center be at z = 0 */
03797 /***** */
03798
03799 l=L/2; /* half of the membrane length */
03800 z_0 = z_low + l; /* center of the membrane */
03801 G=l*eps_w/eps_m*xkappa;
03802 A=-V/2*(1/(G+1))*exp(xkappa*l);
03803 B=V/2;
03804 C=-V/2*eps_w/eps_m*xkappa*(1/(G+1));
03805 D=-A;
03806 /* The analytic expression for the boundary conditions */
03807 /* had the cytoplasmic surface of the membrane set to zero. */
03808 /* This requires an off-set of the BC equations. */

```

```

03809
03810     /* the "i" boundaries (dirichlet) */
03811     for (k=0; k<nz; k++) {
03812         gpos[2] = zf[k];
03813         z_rel = gpos[2] - z_0;      /* relative position for BCs */
03814
03815         for (j=0; j<ny; j++) {
03816
03817             if (gpos[2] <= z_low) {                                /* cytoplasmic */
03818
03819                 val = A*exp(xkappa*z_rel) + V;
03820                 gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
03821                 gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */
03822
03823         }
03824
03825         else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
03826
03827             val = B + C*z_rel;
03828             gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
03829             gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */
03830
03831     }
03832
03833     else if (gpos[2] > z_high) {                            /* extracellular */
03834
03835         val = D*exp(-xkappa*z_rel);
03836         gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
03837         gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */
03838
03839     }
03840
03841     }
03842 }
03843
03844     /* the "j" boundaries (dirichlet) */
03845     for (k=0; k<nz; k++) {
03846         gpos[2] = zf[k];
03847         z_rel = gpos[2] - z_0;
03848         for (i=0; i<nx; i++) {
03849
03850             if (gpos[2] <= z_low) {                                /* cytoplasmic */
03851
03852                 val = A*exp(xkappa*z_rel) + V;
03853                 gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
03854                 gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
03855 //printf("%f \n",val);
03856
03857     }
03858
03859     else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
03860
03861         val = B + C*z_rel;
03862         gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
03863         gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
03864 //printf("%f \n",val);
03865

```

```

03866     }
03867     else if (gpos[2] > z_high) { /* extracellular */
03868
03869     val = D*exp(-xkappa*z_rel);
03870     gycf[IJKy(i,k,0)] += val; /* assign low side BC */
03871     gycf[IJKy(i,k,1)] += val; /* assign high side BC */
03872     //printf("%f \n",val);
03873
03874 }
03875
03876 }
03877 }
03878
03879 /* the "k" boundaries (dirichlet) */
03880 for (j=0; j<ny; j++) {
03881     for (i=0; i<nx; i++) {
03882
03883     /* first assign the bottom boundary */
03884
03885     gpos[2] = zf[0];
03886     z_rel = gpos[2] - z_0;
03887
03888     if (gpos[2] <= z_low) { /* cytoplasmic */
03889
03890     val = A*exp(xkappa*z_rel) + V;
03891     gzcf[IJKz(i,j,0)] += val; /* assign low side BC */
03892     //printf("%f \n",val);
03893
03894 }
03895
03896     else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
03897
03898     val = B + C*z_rel;
03899     gzcf[IJKz(i,j,0)] += val; /* assign low side BC */
03900
03901 }
03902
03903     else if (gpos[2] > z_high) { /* extracellular */
03904
03905     val = D*exp(-xkappa*z_rel);
03906     gzcf[IJKz(i,j,0)] += val; /* assign low side BC */
03907
03908 }
03909
03910     /* now assign the top boundary */
03911
03912     gpos[2] = zf[nz-1];
03913     z_rel = gpos[2] - z_0;
03914
03915     if (gpos[2] <= z_low) { /* cytoplasmic */
03916
03917     val = A*exp(xkappa*z_rel) + V;
03918     gzcf[IJKz(i,j,1)] += val; /* assign high side BC */
03919
03920 }
03921
03922     else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */

```

```

03923
03924     val = B + C*z_rel;
03925     gzcfc[IJKz(i,j,1)] += val;      /* assign high side BC */
03926
03927 }
03928
03929 else if (gpos[2] > z_high) {           /* extracellular */
03930
03931     val = D*exp(-xkappa*z_rel);
03932     gzcfc[IJKz(i,j,1)] += val;      /* assign high side BC */
03933 //printf("%f \n",val);
03934
03935 }
03936
03937 }
03938 }
03939 }
03940
03941 VPUBLIC void bcf1_map(Vpmg *thee) {
03942
03943 Vpbe *pbe;
03944     double position[3], pot, hx, hy, hzed;
03945     int i, j, k, nx, ny, nz, rc;
03946
03947
03948     VASSERT(thee != VNULL);
03949
03950     /* Mesh info */
03951     nx = thee->pmgp->nx;
03952     ny = thee->pmgp->ny;
03953     nz = thee->pmgp->nz;
03954     hx = thee->pmgp->hx;
03955     hy = thee->pmgp->hy;
03956     hzed = thee->pmgp->hzed;
03957
03958     /* Reset the potential array */
03959     for (i=0; i<(nx*ny*nz); i++) thee->pot[i] = 0.0;
03960
03961     /* Fill in the source term (atomic potentials) */
03962     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
03963     for (k=0; k<nz; k++) {
03964         for (j=0; j<ny; j++) {
03965             for (i=0; i<nx; i++) {
03966                 position[0] = thee->xf[i];
03967                 position[1] = thee->yf[j];
03968                 position[2] = thee->zf[k];
03969                 rc = Vgrid_value(thee->potMap, position, &pot);
03970                 if (!rc) {
03971                     Vnm_print(2, "fillcoChargeMap: Error -- fell off of potential map at (%g, %g, %g)\n",
03972                               position[0], position[1], position[2]);
03973                     VASSERT(0);
03974                 }
03975                 thee->pot[IJK(i,j,k)] = pot;
03976             }
03977         }
03978     }

```

```

03979
03980 }
03981
03982 #if defined(WITH_TINKER)
03983 VPRIIVATE void bcfl_mdh_tinker(Vpmg *thee) {
03984
03985 int i,j,k,iatom;
03986 int nx, ny, nz;
03987
03988 double val, *apos, gpos[3], tensor[9];
03989 double *dipole, *quadrupole;
03990 double size, charge, xkappa, eps_w, eps_p, T, prel, dist;
03991
03992 double ux,uy,uz,xr,yr,zr;
03993 double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
03994
03995 double *xf, *yf, *zf;
03996 double *gxcf, *gycf, *gzcf;
03997
03998 Vpbe *pbe;
03999 Vatom *atom;
04000 Valist *alist;
04001
04002 pbe = thee->pbe;
04003 alist = thee->pbe->alist;
04004     nx = thee->pmgp->nx;
04005     ny = thee->pmgp->ny;
04006     nz = thee->pmgp->nz;
04007
04008 xf = thee->xf;
04009 yf = thee->yf;
04010 zf = thee->zf;
04011
04012 gxcf = thee->gxcf;
04013 gycf = thee->gycf;
04014 gzcf = thee->gzcf;
04015
04016 /* For each "atom" (only one for bcfl=1), we use the following formula to
04017 * calculate the boundary conditions:
04018 *      g(x) = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_b T} \exp(-xkappa*(d - a)) / (1 + xkappa*a)
04019 *      * 1/d
04020 * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
04021 * We only need to evaluate some of these prefactors once:
04022 *      prel = \frac{e_c}{4\pi\epsilon_0\epsilon_w k_b T}
04023 * which gives the potential as
04024 *      g(x) = prel * q/d * \frac{\exp(-xkappa*(d - a))}{(1 + xkappa*a)}
04025 */
04026
04027     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
04028     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
04029     T = Vpbe_getTemperature(pbe);                /* K */
04030     prel = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
04031
04032 /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
04033 * m/A, then we will only need to deal with distances and sizes in
04034 * Angstroms rather than meters. */
04035     xkappa = Vpbe_getXkappa(pbe);               /* A^{-1} */

```

```

04036     prel = prel*(1.0e10);
04037
04038 /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
04039 * m/A, then we will only need to deal with distances and sizes in
04040 * Angstroms rather than meters. */  

04041     xkappa = Vpbe_getXkappa(pbe);           /* A^{-1} */  

04042
04043     for(k=0;k<nz;k++) {
04044         gpos[2] = zf[k];
04045         for(j=0;j<ny;j++) {
04046             gpos[1] = yf[j];
04047             for(i=0;i<nx;i++) {
04048                 gpos[0] = xf[i];
04049                 if(gridPointIsValid(i, j, k, nx, ny, nz)) {
04050                     val = 0.0;
04051
04052                     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04053                         atom = Valist_getAtom(alist, iatom);
04054                         apos = Vatom_getPosition(atom);
04055                         size = Vatom_getRadius(atom);
04056
04057                         charge = 0.0;
04058
04059                         dipole = VNULL;
04060                         quadrupole = VNULL;
04061
04062                         if (thee->chargeSrc == VCM_PERMANENT) {
04063                             charge = Vatom_getCharge(atom);
04064                             dipole = Vatom_getDipole(atom);
04065                             quadrupole = Vatom_getQuadrupole(atom);
04066                         } else if (thee->chargeSrc == VCM_INDUCED) {
04067                             dipole = Vatom_getInducedDipole(atom);
04068                         } else {
04069                             dipole = Vatom_getNLInducedDipole(atom);
04070                         }
04071
04072                         ux = dipole[0];
04073                         uy = dipole[1];
04074                         uz = dipole[2];
04075
04076                         if (quadrupole != VNULL) {
04077                             /* The factor of 1/3 results from using a
04078                                traceless quadrupole definition. See, for example,
04079                                "The Theory of Intermolecular Forces" by A.J. Stone,
04080                                Chapter 3. */
04081                             qxx = quadrupole[0] / 3.0;
04082                             qxy = quadrupole[1] / 3.0;
04083                             qxz = quadrupole[2] / 3.0;
04084                             qyx = quadrupole[3] / 3.0;
04085                             qyy = quadrupole[4] / 3.0;
04086                             qyz = quadrupole[5] / 3.0;
04087                             qzx = quadrupole[6] / 3.0;
04088                             qzy = quadrupole[7] / 3.0;
04089                             qzz = quadrupole[8] / 3.0;
04090                         } else {
04091                             qxx = 0.0;
04092

```

```

04093     qxy = 0.0;
04094     qxz = 0.0;
04095     qyx = 0.0;
04096     qyy = 0.0;
04097     qyz = 0.0;
04098     qzx = 0.0;
04099     qzy = 0.0;
04100     qzz = 0.0;
04101 }
04102
04103     xr = gpos[0] - apos[0];
04104     yr = gpos[1] - apos[1];
04105     zr = gpos[2] - apos[2];
04106
04107     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
04108     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
04109
04110     val += prel*charge*tensor[0];
04111     val -= prel*ux*xr*tensor[1];
04112     val -= prel*uy*yr*tensor[1];
04113     val -= prel*uz*zr*tensor[1];
04114     val += prel*qxx*xr*xr*tensor[2];
04115     val += prel*qyy*yryr*tensor[2];
04116     val += prel*qzz*zr*zr*tensor[2];
04117     val += prel*2.0*qxy*xr*yrtensor[2];
04118     val += prel*2.0*qxz*xr*zrtensor[2];
04119     val += prel*2.0*qyz*yr*zrtensor[2];
04120
04121 }
04122
04123     if(i==0) {
04124         gxcf[IJKx(j,k,0)] = val;
04125     }
04126     if(i==nx-1) {
04127         gxcf[IJKx(j,k,1)] = val;
04128     }
04129     if(j==0) {
04130         gycf[IJKy(i,k,0)] = val;
04131     }
04132     if(j==ny-1) {
04133         gycf[IJKy(i,k,1)] = val;
04134     }
04135     if(k==0) {
04136         gzcf[IJKz(i,j,0)] = val;
04137     }
04138     if(k==nz-1) {
04139         gzcf[IJKz(i,j,1)] = val;
04140     }
04141 } /* End grid point is valid */
04142 } /* End i loop */
04143 } /* End j loop */
04144 } /* End k loop */
04145
04146 }
04147 #endif
04148
04149 VPRIIVATE void bcCalc(Vpmg *thee) {

```

```

04150
04151     int i, j, k;
04152     int nx, ny, nz;
04153
04154     double zmem, eps_m, Lmem, memv, eps_w, xkappa;
04155
04156     nx = thee->pmgp->nx;
04157     ny = thee->pmgp->ny;
04158     nz = thee->pmgp->nz;
04159
04160     /* Zero out the boundaries */
04161     /* the "i" boundaries (dirichlet) */
04162     for (k=0; k<nz; k++) {
04163         for (j=0; j<ny; j++) {
04164             thee->gxcf[IJKx(j,k,0)] = 0.0;
04165             thee->gxcf[IJKx(j,k,1)] = 0.0;
04166             thee->gxcf[IJKx(j,k,2)] = 0.0;
04167             thee->gxcf[IJKx(j,k,3)] = 0.0;
04168         }
04169     }
04170
04171     /* the "j" boundaries (dirichlet) */
04172     for (k=0; k<nz; k++) {
04173         for (i=0; i<nx; i++) {
04174             thee->gycf[IJKy(i,k,0)] = 0.0;
04175             thee->gycf[IJKy(i,k,1)] = 0.0;
04176             thee->gycf[IJKy(i,k,2)] = 0.0;
04177             thee->gycf[IJKy(i,k,3)] = 0.0;
04178         }
04179     }
04180
04181     /* the "k" boundaries (dirichlet) */
04182     for (j=0; j<ny; j++) {
04183         for (i=0; i<nx; i++) {
04184             thee->gzcf[IJKz(i,j,0)] = 0.0;
04185             thee->gzcf[IJKz(i,j,1)] = 0.0;
04186             thee->gzcf[IJKz(i,j,2)] = 0.0;
04187             thee->gzcf[IJKz(i,j,3)] = 0.0;
04188         }
04189     }
04190
04191     switch (thee->pmgp->bcfl) {
04192         /* If we have zero boundary conditions, we're done */
04193         case BCFL_ZERO:
04194             return;
04195         case BCFL_SDH:
04196             bcfl_sdh(thee);
04197             break;
04198         case BCFL_MDH:
04199 #if defined(WITH_TINKER)
04200             bcfl_mdh_tinker(thee);
04201 #else
04202 #ifdef DEBUG_MAC OSX OCL
04204 #include "mach_chud.h"
04205             uint64_t mbeg = mach_absolute_time();
04206

```

```

04207     /*
04208      * If OpenCL is available we use it, otherwise fall back to
04209      * normal route (CPU multithreaded w/ OpenMP)
04210      */
04211     if (kOpenCLAvailable == 1) bcflnewOpenCL(thee);
04212     else bcflnew(thee);
04213
04214     mets_(&mbeg, "MDH");
04215 #else
04216     /* bcfl_mdh(thee); */
04217     bcflnew(thee);
04218 #endif /* DEBUG_MAC OSX_OCL */
04219
04220 #endif /* WITH_TINKER */
04221     break;
04222 case BCFL_MEM:
04223
04224     zmem = Vpbe_getzmem(thee->pbe);
04225     Lmem = Vpbe_getLmem(thee->pbe);
04226     eps_m = Vpbe_getmembraneDiel(thee->pbe);
04227     memv = Vpbe_getmemv(thee->pbe);
04228
04229     eps_w = Vpbe_getSolventDiel(thee->pbe);
04230     xkappa = Vpbe_getXkappa(thee->pbe);
04231
04232     bcfl_mem(zmem, Lmem, eps_m, eps_w, memv, xkappa,
04233             thee->gxcf, thee->gycf, thee->gzcf,
04234             thee->xf, thee->yf, thee->zf, nx, ny, nz);
04235     break;
04236 case BCFL_UNUSED:
04237     Vnm_print(2, "bcCalc: Invalid bcfl (%d)!\n", thee->pmgp->bcfl);
04238     VASSERT(0);
04239     break;
04240 case BCFL_FOCUS:
04241     Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
04242     VASSERT(0);
04243     break;
04244 case BCFL_MAP:
04245     bcfl_map(thee);
04246     focusFillBound(thee,VNULL);
04247     break;
04248 default:
04249     Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
04250     flag (%d)!\n", thee->pmgp->bcfl);
04251     VASSERT(0);
04252     break;
04253 }
04254 }
04255
04256 VPRIVATE void fillcoCoefMap(Vpmg *thee) {
04257
04258     Vpbe *pbe;
04259     double ionstr, position[3], tkappa, eps, pot, hx, hy, hzed;
04260     int i, j, k, nx, ny, nz;
04261     double kappamax;
04262     VASSERT(thee != VNULL);
04263

```

```

04264     /* Get PBE info */
04265     pbe = thee->pbe;
04266     ionstr = Vpbe_getBulkIonicStrength(pbe);
04267
04268     /* Mesh info */
04269     nx = thee->pmgp->nx;
04270     ny = thee->pmgp->ny;
04271     nz = thee->pmgp->nz;
04272     hx = thee->pmgp->hx;
04273     hy = thee->pmgp->hy;
04274     hzed = thee->pmgp->hzed;
04275
04276     if ((!thee->useDielXMap) || (!thee->useDielYMap)
04277     || (!thee->useDielZMap) || ((!thee->useKappaMap) && (ionstr>VPMGSMALL))) {
04278
04279         Vnm_print(2, "fillcoCoefMap: You need to use all coefficient maps!\n");
04280         VASSERT(0);
04281     }
04282
04283
04284     /* Scale the kappa map to values between 0 and 1
04285      Thus get the maximum value in the map - this
04286      is theoretically unnecessary, but a good check.*/
04287     kappamax = -1.00;
04288     for (k=0; k<nz; k++) {
04289         for (j=0; j<ny; j++) {
04290             for (i=0; i<nx; i++) {
04291                 if (ionstr > VPMGSMALL) {
04292                     position[0] = thee->xf[i];
04293                     position[1] = thee->yf[j];
04294                     position[2] = thee->zf[k];
04295                     if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
04296                         Vnm_print(2, "Vpmg_fillco: Off kappaMap at:\n");
04297                         Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04298                                 position[0], position[1], position[2]);
04299                         VASSERT(0);
04300                     }
04301                     if (tkappa > kappamax) {
04302                         kappamax = tkappa;
04303                     }
04304                     if (tkappa < 0.0){
04305                         Vnm_print(2, "Vpmg_fillcoCoefMap: Kappa map less than 0\n"
04306                     );
04307                     Vnm_print(2, "Vpmg_fillcoCoefMap: at (x,y,z) = (%g,%g %g)\n"
04308                     );
04309                     position[0], position[1], position[2]);
04310                     VASSERT(0);
04311                 }
04312             }
04313         }
04314
04315     if (kappamax > 1.0){
04316         Vnm_print(2, "Vpmg_fillcoCoefMap: Maximum Kappa value\n");
04317         Vnm_print(2, "%g is greater than 1 - will scale appropriately!\n",
04318         kappamax);

```

```

04319     }
04320     else {
04321         kappamax = 1.0;
04322     }
04323
04324     for (k=0; k<nz; k++) {
04325         for (j=0; j<ny; j++) {
04326             for (i=0; i<nx; i++) {
04327
04328                 if (ionstr > VPMGSMALL) {
04329                     position[0] = thee->xf[i];
04330                     position[1] = thee->yf[j];
04331                     position[2] = thee->zf[k];
04332                     if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
04333                         Vnm_print(2, "Vpmg_fillco: Off kappaMap at:\n");
04334                         Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04335                             position[0], position[1], position[2]);
04336                         VASSERT(0);
04337                     }
04338                     if (tkappa < VPMGSMALL) tkappa = 0.0;
04339                     thee->kappa[IJK(i,j,k)] = (tkappa / kappamax);
04340                 }
04341
04342                 position[0] = thee->xf[i] + 0.5*hx;
04343                 position[1] = thee->yf[j];
04344                 position[2] = thee->zf[k];
04345                 if (!Vgrid_value(thee->dielXMap, position, &eps)) {
04346                     Vnm_print(2, "Vpmg_fillco: Off dielXMap at:\n");
04347                     Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04348                         position[0], position[1], position[2]);
04349                     VASSERT(0);
04350                 }
04351                 thee->epsx[IJK(i,j,k)] = eps;
04352
04353                 position[0] = thee->xf[i];
04354                 position[1] = thee->yf[j] + 0.5*hy;
04355                 position[2] = thee->zf[k];
04356                 if (!Vgrid_value(thee->dielYMap, position, &eps)) {
04357                     Vnm_print(2, "Vpmg_fillco: Off dielYMap at:\n");
04358                     Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04359                         position[0], position[1], position[2]);
04360                     VASSERT(0);
04361                 }
04362                 thee->epsy[IJK(i,j,k)] = eps;
04363
04364                 position[0] = thee->xf[i];
04365                 position[1] = thee->yf[j];
04366                 position[2] = thee->zf[k] + 0.5*hzed;
04367                 if (!Vgrid_value(thee->dielZMap, position, &eps)) {
04368                     Vnm_print(2, "Vpmg_fillco: Off dielZMap at:\n");
04369                     Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04370                         position[0], position[1], position[2]);
04371                     VASSERT(0);
04372                 }
04373                 thee->epsz[IJK(i,j,k)] = eps;
04374             }
04375         }

```

```

04376     }
04377 }
04378
04379 VPRIVATE void fillcoCoefMol(Vpmg *thee) {
04380
04381     if (thee->useDielXMap || thee->useDielYMap || thee->useDielZMap ||
04382         thee->useKappaMap) {
04383
04384         fillcoCoefMap(thee);
04385
04386     } else {
04387
04388         fillcoCoefMolDiel(thee);
04389         fillcoCoefMolIon(thee);
04390
04391     }
04392
04393 }
04394
04395 VPRIVATE void fillcoCoefMolIon(Vpmg *thee) {
04396
04397     Vacc *acc;
04398     Valist *alist;
04399     Vpbe *pbe;
04400     Vatom *atom;
04401     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr;
04402     double xlen, ylen, zlen, irad;
04403     double hx, hy, hzed, *apos, arad;
04404     int i, nx, ny, nz, iatom;
04405     Vsurf_Meth surfMeth;
04406
04407     VASSERT(thee != VNULL);
04408     surfMeth = thee->surfMeth;
04409
04410     /* Get PBE info */
04411     pbe = thee->pbe;
04412     acc = pbe->acc;
04413     alist = pbe->alist;
04414     irad = Vpbe_getMaxIonRadius(pbe);
04415     ionstr = Vpbe_getBulkIonicStrength(pbe);
04416
04417     /* Mesh info */
04418     nx = thee->pmgp->nx;
04419     ny = thee->pmgp->ny;
04420     nz = thee->pmgp->nz;
04421     hx = thee->pmgp->hx;
04422     hy = thee->pmgp->hy;
04423     hzed = thee->pmgp->hzed;
04424
04425     /* Define the total domain size */
04426     xlen = thee->pmgp->xlen;
04427     ylen = thee->pmgp->ylen;
04428     zlen = thee->pmgp->zlen;
04429
04430     /* Define the min/max dimensions */
04431     xmin = thee->pmgp->xcent - (xlen/2.0);
04432     ymin = thee->pmgp->ycent - (ylen/2.0);

```

```

04433     zmin = thee->pmgp->zcent - (zlen/2.0);
04434     xmax = thee->pmgp->xcent + (xlen/2.0);
04435     ymax = thee->pmgp->ycent + (ylen/2.0);
04436     zmax = thee->pmgp->zcent + (zlen/2.0);
04437
04438     /* This is a floating point parameter related to the non-zero nature of the
04439      * bulk ionic strength. If the ionic strength is greater than zero; this
04440      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
04441      * Otherwise, this parameter is set to 0.0 */
04442     if (ionstr > VPMGSMALL) ionmask = 1.0;
04443     else ionmask = 0.0;
04444
04445     /* Reset the kappa array, marking everything accessible */
04446     for (i=0; i<(nx*ny*nz); i++) thee->kappa[i] = ionmask;
04447
04448     if (ionstr < VPMGSMALL) return;
04449
04450     /* Loop through the atoms and set kappa = 0.0 (inaccessible) if a point
04451      * is inside the ion-inflated van der Waals radii */
04452     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04453
04454         atom = Valist_getAtom(alist, iatom);
04455         apos = Vatom_getPosition(atom);
04456         arad = Vatom_getRadius(atom);
04457
04458         if (arad > VSMALL) {
04459
04460             /* Make sure we're on the grid */
04461             if ((apos[0]<(xmin-irad-arad)) || (apos[0]>(xmax+irad+arad)) || \
04462                 (apos[1]<(ymin-irad-arad)) || (apos[1]>(ymax+irad+arad)) || \
04463                 (apos[2]<(zmin-irad-arad)) || (apos[2]>(zmax+irad+arad))) {
04464                 if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
04465                     (thee->pmgp->bcfl != BCFL_MAP)) {
04466                     Vnm_print(2,
04467                         "Vpmg_fillco: Atom #%"PRIu32" at (%f, %f, %f) is off the mesh (ignoring):\n",
04468                         iatom, apos[0], apos[1], apos[2]);
04469                     Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
04470                         xmin, xmax);
04471                     Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
04472                         ymin, ymax);
04473                     Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
04474                         zmin, zmax);
04475                 }
04476                 fflush(stderr);
04477
04478             } else { /* if we're on the mesh */
04479
04480                 /* Mark ions */
04481                 markSphere((irad+arad), apos,
04482                             nx, ny, nz,
04483                             hx, hy, hzed,
04484                             xmin, ymin, zmin,
04485                             thee->kappa, 0.0);
04486
04487             } /* endif (on the mesh) */
04488         }

```

```

04489     } /* endfor (over all atoms) */
04490
04491 }
04492
04493 VPRIVATE void fillcoCoefMolDiel(Vpmg *thee) {
04494
04495     /* Always call NoSmooth to fill the epsilon arrays */
04496     fillcoCoefMolDielNoSmooth(thee);
04497
04498     /* Call the smoothing algorithm as needed */
04499     if (thee->surfMeth == VSM_MOLSMOOTH) {
04500         fillcoCoefMolDielSmooth(thee);
04501     }
04502 }
04503
04504 VPRIVATE void fillcoCoefMolDielNoSmooth(Vpmg *thee) {
04505
04506     Vacc *acc;
04507     VaccSurf *asurf;
04508     Valist *alist;
04509     Vpbe *pbe;
04510     Vatom *atom;
04511     double xmin, xmax, ymin, ymax, zmin, zmax;
04512     double xlabel, ylabel, zlabel, position[3];
04513     double srad, epsw, epsp, deps, area;
04514     double hx, hy, hzed, *apos, arad;
04515     int i, nx, ny, nz, ntot, iatom, ipt;
04516
04517     /* Get PBE info */
04518     pbe = thee->pbe;
04519     acc = pbe->acc;
04520     alist = pbe->alist;
04521     srad = Vpbe_getSolventRadius(pbe);
04522     epsw = Vpbe_getSolventDiel(pbe);
04523     epsp = Vpbe_getSoluteDiel(pbe);
04524
04525     /* Mesh info */
04526     nx = thee->pmgp->nx;
04527     ny = thee->pmgp->ny;
04528     nz = thee->pmgp->nz;
04529     xlabel = thee->pmgp->hx;
04530     ylabel = thee->pmgp->hy;
04531     hzed = thee->pmgp->hzed;
04532
04533     /* Define the total domain size */
04534     xlabel = thee->pmgp->xlabel;
04535     ylabel = thee->pmgp->ylabel;
04536     zlabel = thee->pmgp->zlabel;
04537
04538     /* Define the min/max dimensions */
04539     xmin = thee->pmgp->xcent - (xlabel/2.0);
04540     ymin = thee->pmgp->ycent - (ylabel/2.0);
04541     zmin = thee->pmgp->zcent - (zlabel/2.0);
04542     xmax = thee->pmgp->xcent + (xlabel/2.0);
04543     ymax = thee->pmgp->ycent + (ylabel/2.0);
04544     zmax = thee->pmgp->zcent + (zlabel/2.0);
04545

```

```

04546     /* Reset the arrays */
04547     ntot = nx*ny*nz;
04548     for (i=0; i<ntot; i++) {
04549         thee->epsx[i] = epsw;
04550         thee->epsy[i] = epsw;
04551         thee->epsz[i] = epsw;
04552     }
04553
04554     /* Loop through the atoms and set a{123}cf = 0.0 (inaccessible)
04555      * if a point is inside the solvent-inflated van der Waals radii */
04556 #pragma omp parallel for default(shared) private(iatom,atom,apos,arad)
04557     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04558
04559         atom = Valist_getAtom(alist, iatom);
04560         apos = Vatom_getPosition(atom);
04561         arad = Vatom_getRadius(atom);
04562
04563         /* Make sure we're on the grid */
04564         if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
04565             (apos[1]<=ymin) || (apos[1]>=ymax) || \
04566             (apos[2]<=zmin) || (apos[2]>=zmax)) {
04567             if ((thee->pmgp->bclf != BCFL_FOCUS) &&
04568                 (thee->pmgp->bclf != BCFL_MAP)) {
04569                 Vnm_print(2, "Vpmg_fillco: Atom %d at (%.3f, %.3f,\n"
04570                 "%.3f) is off the mesh (ignoring):\n",
04571                 iatom, apos[0], apos[1], apos[2]);
04572                 Vnm_print(2, "Vpmg_fillco: xmin = %.g, xmax = %.g\n",
04573                 xmin, xmax);
04574                 Vnm_print(2, "Vpmg_fillco: ymin = %.g, ymax = %.g\n",
04575                 ymin, ymax);
04576                 Vnm_print(2, "Vpmg_fillco: zmin = %.g, zmax = %.g\n",
04577                 zmin, zmax);
04578             }
04579             fflush(stderr);
04580
04581         } else { /* if we're on the mesh */
04582
04583             if (arad > VSMALL) {
04584                 /* Mark x-shifted dielectric */
04585                 markSphere((arad+srad), apos,
04586                             nx, ny, nz,
04587                             hx, hy, hzed,
04588                             (xmin+0.5*hx), ymin, zmin,
04589                             thee->epsx, epsp);
04590
04591                 /* Mark y-shifted dielectric */
04592                 markSphere((arad+srad), apos,
04593                             nx, ny, nz,
04594                             hx, hy, hzed,
04595                             xmin, (ymin+0.5*hy), zmin,
04596                             thee->epsy, epsp);
04597
04598                 /* Mark z-shifted dielectric */
04599                 markSphere((arad+srad), apos,
04600                             nx, ny, nz,
04601                             hx, hy, hzed,
04602                             xmin, ymin, (zmin+0.5*hzed),

```

```

04603                     thee->epsz, epsp);
04604                 }
04605
04606             } /* endif (on the mesh) */
04607         } /* endfor (over all atoms) */
04608
04609     area = Vacc_SASA(acc, srad);
04610
04611     /* We only need to do the next step for non-zero solvent radii */
04612     if (srad > VSMALL) {
04613
04614         /* Now loop over the solvent accessible surface points */
04615
04616 #pragma omp parallel for default(shared) private(iatom,atom,area,asurf,ipt,positi
04617 on)
04617     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04618         atom = Valist_getAtom(alist, iatom);
04619         area = Vacc_atomSASA(acc, srad, atom);
04620         if (area > 0.0) {
04621             asurf = Vacc_atomSASPoints(acc, srad, atom);
04622
04623             /* Use each point on the SAS to reset the solvent accessibility */
04624             /* TODO: Make sure we're not still wasting time here. */
04625             for (ipt=0; ipt<(asurf->npts); ipt++) {
04626
04627                 position[0] = asurf->xpts[ipt];
04628                 position[1] = asurf->ypts[ipt];
04629                 position[2] = asurf->zpts[ipt];
04630
04631                 /* Mark x-shifted dielectric */
04632                 markSphere(srad, position,
04633                     nx, ny, nz,
04634                     hx, hy, hzed,
04635                     (xmin+0.5*hx), ymin, zmin,
04636                     thee->epsx, epsw);
04637
04638                 /* Mark y-shifted dielectric */
04639                 markSphere(srad, position,
04640                     nx, ny, nz,
04641                     hx, hy, hzed,
04642                     xmin, (ymin+0.5*hy), zmin,
04643                     thee->epsy, epsw);
04644
04645                 /* Mark z-shifted dielectric */
04646                 markSphere(srad, position,
04647                     nx, ny, nz,
04648                     hx, hy, hzed,
04649                     xmin, ymin, (zmin+0.5*hzed),
04650                     thee->epsz, epsw);
04651
04652             }
04653         }
04654     }
04655 }
04656 }
04657
04658 VPRIvate void fillcoCoefMolDielsSmooth(Vpmg *thee) {

```

```

04659
04660    /* This function smoothes using a 9 point method based on
04661        Brucoleri, et al. J Comput Chem 18 268-276 (1997). The nine points
04662        used are the shifted grid point and the 8 points that are 1/sqrt(2)
04663        grid spacings away. The harmonic mean of the 9 points is then used to
04664        find the overall dielectric value for the point in question. The use of
04665        this function assumes that the non-smoothed values were placed in the
04666        dielectric arrays by the fillCoefMolDielsSmooth function.*/
04667
04668    Vpbe *pbe;
04669    double frac, epsw;
04670    int i, j, k, nx, ny, nz, numpts;
04671
04672    /* Mesh info */
04673    nx = thee->pmgp->nx;
04674    ny = thee->pmgp->ny;
04675    nz = thee->pmgp->nz;
04676
04677    pbe = thee->pbe;
04678    epsw = Vpbe_getSolventDiels(pbe);
04679
04680    /* Copy the existing diel arrays to work arrays */
04681    for (i=0; i<(nx*ny*nz); i++) {
04682        thee->a1cf[i] = thee->epsx[i];
04683        thee->a2cf[i] = thee->epsy[i];
04684        thee->a3cf[i] = thee->epsz[i];
04685        thee->epsx[i] = epsw;
04686        thee->epsy[i] = epsw;
04687        thee->epsz[i] = epsw;
04688    }
04689
04690    /* Smooth the dielectric values */
04691    for (i=0; i<nx; i++) {
04692        for (j=0; j<ny; j++) {
04693            for (k=0; k<nz; k++) {
04694
04695                /* Get the 8 points that are 1/sqrt(2) grid spacings away */
04696
04697                /* Points for the X-shifted array */
04698                frac = 1.0/thee->a1cf[IJK(i,j,k)];
04699                frac += 1.0/thee->a2cf[IJK(i,j,k)];
04700                frac += 1.0/thee->a3cf[IJK(i,j,k)];
04701                numpts = 3;
04702
04703                if (j > 0) {
04704                    frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
04705                    numpts += 1;
04706                }
04707                if (k > 0) {
04708                    frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04709                    numpts += 1;
04710                }
04711                if (i < (nx-1)){
04712                    frac += 1.0/thee->a2cf[IJK(i+1,j,k)];
04713                    frac += 1.0/thee->a3cf[IJK(i+1,j,k)];
04714                    numpts += 2;
04715                    if (j > 0) {

```

```

04716         frac += 1.0/thee->a2cf[IJK(i+1,j-1,k)];
04717         numpts += 1;
04718     }
04719     if (k > 0) {
04720         frac += 1.0/thee->a3cf[IJK(i+1,j,k-1)];
04721         numpts += 1;
04722     }
04723 }
04724 thee->epsx[IJK(i,j,k)] = numpts/frac;
04725
/* Points for the Y-shifted array */
04726 frac = 1.0/thee->a2cf[IJK(i,j,k)];
04727 frac += 1.0/thee->a1cf[IJK(i,j,k)];
04728 frac += 1.0/thee->a3cf[IJK(i,j,k)];
04729 numpts = 3;
04730
04731 if (i > 0) {
04732     frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
04733     numpts += 1;
04734 }
04735 if (k > 0) {
04736     frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04737     numpts += 1;
04738 }
04739 if (j < (ny-1)){
04740     frac += 1.0/thee->a1cf[IJK(i,j+1,k)];
04741     frac += 1.0/thee->a3cf[IJK(i,j+1,k)];
04742     numpts += 2;
04743     if (i > 0) {
04744         frac += 1.0/thee->a1cf[IJK(i-1,j+1,k)];
04745         numpts += 1;
04746     }
04747     if (k > 0) {
04748         frac += 1.0/thee->a3cf[IJK(i,j+1,k-1)];
04749         numpts += 1;
04750     }
04751 }
04752 }
04753 thee->epsy[IJK(i,j,k)] = numpts/frac;
04754
/* Points for the Z-shifted array */
04755 frac = 1.0/thee->a3cf[IJK(i,j,k)];
04756 frac += 1.0/thee->a1cf[IJK(i,j,k)];
04757 frac += 1.0/thee->a2cf[IJK(i,j,k)];
04758 numpts = 3;
04759
04760 if (i > 0) {
04761     frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
04762     numpts += 1;
04763 }
04764 if (j > 0) {
04765     frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
04766     numpts += 1;
04767 }
04768 if (k < (nz-1)){
04769     frac += 1.0/thee->a1cf[IJK(i,j,k+1)];
04770     frac += 1.0/thee->a2cf[IJK(i,j,k+1)];
04771     numpts += 2;
04772 }
```

```

04773         if (i > 0) {
04774             frac += 1.0/thee->a1cf[IJK(i-1,j,k+1)];
04775             numpts += 1;
04776         }
04777         if (j > 0) {
04778             frac += 1.0/thee->a2cf[IJK(i,j-1,k+1)];
04779             numpts += 1;
04780         }
04781     }
04782     thee->epsz[IJK(i,j,k)] = numpts/frac;
04783 }
04784 }
04785 }
04786 }
04787
04788
04789 VPRIVATE void fillcoCoefSpline(Vpmg *thee) {
04790
04791     Valist *alist;
04792     Vpbe *pbe;
04793     Vatom *atom;
04794     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
04795     double xlabel, ylabel, zlabel, position[3], itot, stot, ictot, ictot2, sctot;
04796     double irad, dx, dy, dz, epsw, epsp, w2i;
04797     double hx, hy, hzed, *apos, arad, sctot2;
04798     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin, w3i;
04799     double dist, value, sm, sm2;
04800     int i, j, k, nx, ny, nz, iatom;
04801     int imin, imax, jmin, jmax, kmin, kmax;
04802
04803     VASSERT(thee != VNULL);
04804     splineWin = thee->splineWin;
04805     w2i = 1.0/(splineWin*splineWin);
04806     w3i = 1.0/(splineWin*splineWin*splineWin);
04807
04808     /* Get PBE info */
04809     pbe = thee->pbe;
04810     alist = pbe->alist;
04811     irad = Vpbe_getMaxIonRadius(pbe);
04812     ionstr = Vpbe_getBulkIonicStrength(pbe);
04813     epsw = Vpbe_getSolventDiel(pbe);
04814     epsp = Vpbe_getSoluteDiel(pbe);
04815
04816     /* Mesh info */
04817     nx = thee->pmgp->nx;
04818     ny = thee->pmgp->ny;
04819     nz = thee->pmgp->nz;
04820     hx = thee->pmgp->hx;
04821     hy = thee->pmgp->hy;
04822     hzed = thee->pmgp->hzed;
04823
04824     /* Define the total domain size */
04825     xlabel = thee->pmgp->xlabel;
04826     ylabel = thee->pmgp->ylabel;
04827     zlabel = thee->pmgp->zlabel;
04828
04829     /* Define the min/max dimensions */

```

```

04830     xmin = thee->pmgp->xcent - (xlen/2.0);
04831     ymin = thee->pmgp->ycent - (ylen/2.0);
04832     zmin = thee->pmgp->zcent - (zlen/2.0);
04833     xmax = thee->pmgp->xcent + (xlen/2.0);
04834     ymax = thee->pmgp->ycent + (ylen/2.0);
04835     zmax = thee->pmgp->zcent + (zlen/2.0);
04836
04837     /* This is a floating point parameter related to the non-zero nature of the
04838      * bulk ionic strength. If the ionic strength is greater than zero; this
04839      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
04840      * Otherwise, this parameter is set to 0.0 */
04841     if (ionstr > VPMGSMALL) ionmask = 1.0;
04842     else ionmask = 0.0;
04843
04844     /* Reset the kappa, epsx, epsy, and epsz arrays */
04845     for (i=0; i<(nx*ny*nz); i++) {
04846         thee->kappa[i] = 1.0;
04847         thee->epsx[i] = 1.0;
04848         thee->epsy[i] = 1.0;
04849         thee->epsz[i] = 1.0;
04850     }
04851
04852     /* Loop through the atoms and do assign the dielectric */
04853     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04854
04855         atom = Valist_getAtom(alist, iatom);
04856         apos = Vatom_getPosition(atom);
04857         arad = Vatom_getRadius(atom);
04858
04859         /* Make sure we're on the grid */
04860         if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
04861             (apos[1]<=ymin) || (apos[1]>=ymax) || \
04862             (apos[2]<=zmin) || (apos[2]>=zmax)) {
04863             if (((thee->pmgp->bclf != BCFL_FOCUS) &&
04864                  (thee->pmgp->bclf != BCFL_MAP)) {
04865                 Vnm_print(2, "Vpmg_fillco: Atom %d at (%4.3f, %4.3f,\n
04866 %4.3f) is off the mesh (ignoring):\n",
04867                         iatom, apos[0], apos[1], apos[2]);
04868                 Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
04869                         xmin, xmax);
04870                 Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
04871                         ymin, ymax);
04872                 Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
04873                         zmin, zmax);
04874             }
04875             fflush(stderr);
04876
04877         } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
04878
04879             /* Convert the atom position to grid reference frame */
04880             position[0] = apos[0] - xmin;
04881             position[1] = apos[1] - ymin;
04882             position[2] = apos[2] - zmin;
04883
04884             /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
04885              * ASSIGNMENT (Steps #1-3) */
04886             itot = irad + arad + splineWin;

```

```

04887     itot2 = VSQR(itot);
04888     ictot = VMAX2(0, (irad + arad - splineWin));
04889     ictot2 = VSQR(ictot);
04890     stot = arad + splineWin;
04891     stot2 = VSQR(stot);
04892     sctot = VMAX2(0, (arad - splineWin));
04893     sctot2 = VSQR(sctot);
04894
04895     /* We'll search over grid points which are in the greater of
04896      * these two radii */
04897     rtot = VMAX2(itot, stot);
04898     rtot2 = VMAX2(itot2, stot2);
04899     dx = rtot + 0.5*hx;
04900     dy = rtot + 0.5*hy;
04901     dz = rtot + 0.5*hzed;
04902     imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
04903     imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
04904     jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
04905     jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
04906     kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
04907     kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
04908     for (i=iimin; i<=imax; i++) {
04909         dx2 = VSQR(position[0] - hx*i);
04910         for (j=jmin; j<=jmax; j++) {
04911             dy2 = VSQR(position[1] - hy*j);
04912             for (k=kmin; k<=kmax; k++) {
04913                 dz2 = VSQR(position[2] - k*hzed);
04914
04915                 /* ASSIGN CCF */
04916                 if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
04917                     dist2 = dz2 + dy2 + dx2;
04918                     if (dist2 >= itot2) {
04919                         ;
04920                     }
04921                     if (dist2 <= ictot2) {
04922                         thee->kappa[IJK(i,j,k)] = 0.0;
04923                     }
04924                     if ((dist2 < itot2) && (dist2 > ictot2)) {
04925                         dist = VSQRT(dist2);
04926                         sm = dist - (arad + irad) + splineWin;
04927                         sm2 = VSQR(sm);
04928                         value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
04929                         thee->kappa[IJK(i,j,k)] *= value;
04930                     }
04931                 }
04932
04933                 /* ASSIGN A1CF */
04934                 if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
04935                     dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
04936                     if (dist2 >= stot2) {
04937                         thee->epsx[IJK(i,j,k)] *= 1.0;
04938                     }
04939                     if (dist2 <= sctot2) {
04940                         thee->epsx[IJK(i,j,k)] = 0.0;
04941                     }
04942                     if ((dist2 > sctot2) && (dist2 < stot2)) {
04943                         dist = VSQRT(dist2);

```

```

04944                     sm = dist - arad + splineWin;
04945                     sm2 = VSQR(sm);
04946                     value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
04947                     thee->epsx[IJK(i,j,k)] *= value;
04948                 }
04949             }
04950
04951             /* ASSIGN A2CF */
04952             if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
04953                 dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
04954                 if (dist2 >= stot2) {
04955                     thee->epsy[IJK(i,j,k)] *= 1.0;
04956                 }
04957                 if (dist2 <= sctot2) {
04958                     thee->epsy[IJK(i,j,k)] = 0.0;
04959                 }
04960                 if ((dist2 > sctot2) && (dist2 < stot2)) {
04961                     dist = VSQRT(dist2);
04962                     sm = dist - arad + splineWin;
04963                     sm2 = VSQR(sm);
04964                     value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
04965                     thee->epsy[IJK(i,j,k)] *= value;
04966                 }
04967             }
04968
04969             /* ASSIGN A3CF */
04970             if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
04971                 dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
04972                 if (dist2 >= stot2) {
04973                     thee->epsz[IJK(i,j,k)] *= 1.0;
04974                 }
04975                 if (dist2 <= sctot2) {
04976                     thee->epsz[IJK(i,j,k)] = 0.0;
04977                 }
04978                 if ((dist2 > sctot2) && (dist2 < stot2)) {
04979                     dist = VSQRT(dist2);
04980                     sm = dist - arad + splineWin;
04981                     sm2 = VSQR(sm);
04982                     value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
04983                     thee->epsz[IJK(i,j,k)] *= value;
04984                 }
04985             }
04986
04987             } /* k loop */
04988         } /* j loop */
04989     } /* i loop */
04990 } /* endif (on the mesh) */
04991 } /* endfor (over all atoms) */
04992
04993 Vnm_print(0, "Vpmg_fillco: filling coefficient arrays\n");
04994 /* Interpret markings and fill the coefficient arrays */
04995 for (k=0; k<nz; k++) {
04996     for (j=0; j<ny; j++) {
04997         for (i=0; i<nx; i++) {
04998             thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
04999
05000

```

```

05001             thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
05002                     + epsp;
05003             thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
05004                     + epsp;
05005             thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
05006                     + epsp;
05007
05008             } /* i loop */
05009         } /* j loop */
05010     } /* k loop */
05011 }
05012 }
05013
05014 VPRIVATE void fillcoCoef(Vpmg *thee) {
05015
05016     VASSERT(thee != VNULL);
05017
05018     if (thee->useDielXMap || thee->useDielYMap ||
05019     thee->useDielZMap || thee->useKappaMap) {
05020         fillcoCoefMap(thee);
05021         return;
05022     }
05023
05024     switch(thee->surfMeth) {
05025         case VSM_MOL:
05026             Vnm_print(0, "fillcoCoef: Calling fillcoCoefMol...\n");
05027             fillcoCoefMol(thee);
05028             break;
05029         case VSM_MOLSMOOTH:
05030             Vnm_print(0, "fillcoCoef: Calling fillcoCoefMol...\n");
05031             fillcoCoefMol(thee);
05032             break;
05033         case VSM_SPLINE:
05034             Vnm_print(0, "fillcoCoef: Calling fillcoCoefSpline...\n");
05035             fillcoCoefSpline(thee);
05036             break;
05037         case VSM_SPLINE3:
05038             Vnm_print(0, "fillcoCoef: Calling fillcoCoefSpline3...\n");
05039             fillcoCoefSpline3(thee);
05040             break;
05041         case VSM_SPLINE4:
05042             Vnm_print(0, "fillcoCoef: Calling fillcoCoefSpline4...\n");
05043             fillcoCoefSpline4(thee);
05044             break;
05045         default:
05046             Vnm_print(2, "fillcoCoef: Invalid surfMeth (%d)!\n",
05047                         thee->surfMeth);
05048             VASSERT(0);
05049             break;
05050     }
05051 }
05052
05053
05054 VPRIVATE Vrc_Codes fillcoCharge(Vpmg *thee) {
05055
05056     Vrc_Codes rc;
05057

```

```

05058     VASSERT(thee != VNULL);
05059
05060     if (thee->useChargeMap) {
05061         return fillcoChargeMap(thee);
05062     }
05063
05064     switch(thee->chargeMeth) {
05065         case VCM_TRIL:
05066             Vnm_print(0, "fillcoCharge: Calling fillcoChargeSpline1...\n");
05067             fillcoChargeSpline1(thee);
05068             break;
05069         case VCM_BSPL2:
05070             Vnm_print(0, "fillcoCharge: Calling fillcoChargeSpline2...\n");
05071             fillcoChargeSpline2(thee);
05072             break;
05073         case VCM_BSPL4:
05074             switch (thee->chargeSrc) {
05075                 case VCM_CHARGE:
05076                     Vnm_print(0, "fillcoCharge: Calling fillcoPermanentMultipole.
..\\n");
05077                     fillcoPermanentMultipole(thee);
05078                     break;
05079 #if defined(WITH_TINKER)
05080                 case VCM_PERMANENT:
05081                     Vnm_print(0, "fillcoCharge: Calling fillcoPermanentMultipole.
..\\n");
05082                     fillcoPermanentMultipole(thee);
05083                     break;
05084                 case VCM_INDUCED:
05085                     Vnm_print(0, "fillcoCharge: Calling fillcoInducedDipole...\n");
05086                     fillcoInducedDipole(thee);
05087                     break;
05088                 case VCM_NLINDUCED:
05089                     Vnm_print(0, "fillcoCharge: Calling fillcoNLInducedDipole...
\\n");
05090                     fillcoNLInducedDipole(thee);
05091                     break;
05092 #endif /* if defined(WITH_TINKER) */
05093             default:
05094                 Vnm_print(2, "fillcoCharge: Invalid chargeSource (%d)!\n",
05095                         thee->chargeSrc);
05096                 return VRC_FAILURE;
05097                 break;
05098             }
05099             break;
05100     default:
05101         Vnm_print(2, "fillcoCharge: Invalid chargeMeth (%d)!\n",
05102                         thee->chargeMeth);
05103         return VRC_FAILURE;
05104         break;
05105     }
05106
05107     return VRC_SUCCESS;
05108 }
05109
05110 VPRIPRIVATE Vrc_Codes fillcoChargeMap(Vpmg *thee) {

```

```

05111
05112     Vpbe *pbe;
05113     double position[3], charge, zmagic, hx, hy, hzed;
05114     int i, j, k, nx, ny, nz, rc;
05115
05116
05117     VASSERT(thee != VNULL);
05118
05119     /* Get PBE info */
05120     pbe = thee->pbe;
05121     zmagic = Vpbe_getZmagic(pbe);
05122
05123     /* Mesh info */
05124     nx = thee->pmgp->nx;
05125     ny = thee->pmgp->ny;
05126     nz = thee->pmgp->nz;
05127     hx = thee->pmgp->hx;
05128     hy = thee->pmgp->hy;
05129     hzed = thee->pmgp->hzed;
05130
05131     /* Reset the charge array */
05132     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05133
05134     /* Fill in the source term (atomic charges) */
05135     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05136     for (k=0; k<nz; k++) {
05137         for (j=0; j<ny; j++) {
05138             for (i=0; i<nx; i++) {
05139                 position[0] = thee->xf[i];
05140                 position[1] = thee->yf[j];
05141                 position[2] = thee->zf[k];
05142                 rc = Vgrid_value(thee->chargeMap, position, &charge);
05143                 if (!rc) {
05144                     Vnm_print(2, "fillcoChargeMap: Error -- fell off of charge map at (%g, %g,
05145 %g)\n",
05146                     position[0], position[1], position[2]);
05147                     return VRC_FAILURE;
05148                 }
05149                 /* Scale the charge to internal units */
05150                 charge = charge*zmagic;
05151                 thee->charge[IJK(i,j,k)] = charge;
05152             }
05153         }
05154     }
05155     return VRC_SUCCESS;
05156 }
05157
05158 VPRIVATE void fillcoChargeSpline1(Vpmg *thee) {
05159     Valist *alist;
05160     Vpbe *pbe;
05161     Vatom *atom;
05162     double xmin, xmax, ymin, ymax, zmin, zmax;
05163     double xlabel, ylabel, zlabel, position[3], ifloat, jfloat, kfloat;
05164     double charge, dx, dy, dz, zmagic, hx, hy, hzed, *apos;
05165     int i, nx, ny, nz, iatom, ihi, ilo, jhi, jlo, khi, klo;

```

```

05167
05168
05169     VASSERT(thee != VNULL);
05170
05171     /* Get PBE info */
05172     pbe = thee->pbe;
05173     alist = pbe->alist;
05174     zmagic = Vpbe_getZmagic(pbe);
05175
05176     /* Mesh info */
05177     nx = thee->pmgp->nx;
05178     ny = thee->pmgp->ny;
05179     nz = thee->pmgp->nz;
05180     hx = thee->pmgp->hx;
05181     hy = thee->pmgp->hy;
05182     hzed = thee->pmgp->hzed;
05183
05184     /* Define the total domain size */
05185     xlen = thee->pmgp->xlen;
05186     ylen = thee->pmgp->ylen;
05187     zlen = thee->pmgp->zlen;
05188
05189     /* Define the min/max dimensions */
05190     xmin = thee->pmgp->xcent - (xlen/2.0);
05191     ymin = thee->pmgp->ycent - (ylen/2.0);
05192     zmin = thee->pmgp->zcent - (zlen/2.0);
05193     xmax = thee->pmgp->xcent + (xlen/2.0);
05194     ymax = thee->pmgp->ycent + (ylen/2.0);
05195     zmax = thee->pmgp->zcent + (zlen/2.0);
05196
05197     /* Reset the charge array */
05198     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05199
05200     /* Fill in the source term (atomic charges) */
05201     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05202     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05203
05204         atom = Valist_getAtom(alist, iatom);
05205         apos = VatomGetPosition(atom);
05206         charge = Vatom_getCharge(atom);
05207
05208         /* Make sure we're on the grid */
05209         if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
05210             (apos[1]<=ymin) || (apos[1]>=ymax) || \
05211             (apos[2]<=zmin) || (apos[2]>=zmax)) {
05212             if ((thee->pmgp->bclf != BCFL_FOCUS) &&
05213                 (thee->pmgp->bclf != BCFL_MAP)) {
05214                 Vnm_print(2, "Vpmg_fillco: Atom #%"PRIu32" at (%4.3f, %4.3f, \
05215 %4.3f) is off the mesh (ignoring):\n",
05216                         iatom, apos[0], apos[1], apos[2]);
05217                 Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
05218                         xmin, xmax);
05219                 Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
05220                         ymin, ymax);
05221                 Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
05222                         zmin, zmax);
05223             }

```

```

05224     fflush(stderr);
05225 } else {
05226
05227     /* Convert the atom position to grid reference frame */
05228     position[0] = apos[0] - xmin;
05229     position[1] = apos[1] - ymin;
05230     position[2] = apos[2] - zmin;
05231
05232     /* Scale the charge to be a delta function */
05233     charge = charge*zmagic/(hx*hy*hzed);
05234
05235     /* Figure out which vertices we're next to */
05236     ifloat = position[0]/hx;
05237     jfloat = position[1]/hy;
05238     kfloat = position[2]/hzed;
05239
05240     ihi = (int)ceil(ifloat);
05241     ilo = (int)floor(ifloat);
05242     jhi = (int)ceil(jfloat);
05243     jlo = (int)floor(jfloat);
05244     khi = (int)ceil(kfloat);
05245     klo = (int)floor(kfloat);
05246
05247     /* Now assign fractions of the charge to the nearby verts */
05248     dx = ifloat - (double)(ilo);
05249     dy = jfloat - (double)(jlo);
05250     dz = kfloat - (double)(klo);
05251     thee->charge[IJK(ihi,jhi,khi)] += (dx*dy*dz*charge);
05252     thee->charge[IJK(ihi,jlo,khi)] += (dx*(1.0-dy)*dz*charge);
05253     thee->charge[IJK(ihi,jhi,klo)] += (dx*dy*(1.0-dz)*charge);
05254     thee->charge[IJK(ihi,jlo,klo)] += (dx*(1.0-dy)*(1.0-dz)*charge);
05255     thee->charge[IJK(ilo,jhi,khi)] += ((1.0-dx)*dy*dz *charge);
05256     thee->charge[IJK(ilo,jlo,khi)] += ((1.0-dx)*(1.0-dy)*dz *charge);
05257     thee->charge[IJK(ilo,jhi,klo)] += ((1.0-dx)*dy*(1.0-dz)*charge);
05258     thee->charge[IJK(ilo,jlo,klo)] += ((1.0-dx)*(1.0-dy)*(1.0-dz)*charge)
05259 ;
05260     } /* endif (on the mesh) */
05261 } /* endfor (each atom) */
05262
05263 VPRIIVATE double bspline2(double x) {
05264
05265     double m2m, m2, m3;
05266
05267     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05268     else m2m = 0.0;
05269     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05270     else m2 = 0.0;
05271
05272     if ((x >= 0.0) && (x <= 3.0)) m3 = 0.5*x*m2m + 0.5*(3.0-x)*m2;
05273     else m3 = 0.0;
05274
05275     return m3;
05276
05277 }
05278
05279 VPRIIVATE double dbspline2(double x) {

```

```
05280
05281     double m2m, m2, dm3;
05282
05283     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05284     else m2m = 0.0;
05285     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05286     else m2 = 0.0;
05287
05288     dm3 = m2m - m2;
05289
05290     return dm3;
05291
05292 }
05293
05294
05295 VPRIVATE void fillcoChargeSpline2(Vpmg *thee) {
05296
05297     Valist *alist;
05298     Vpbe *pbe;
05299     Vatom *atom;
05300     double xmin, xmax, ymin, ymax, zmin, zmax, zmagic;
05301     double xlabel, ylabel, zlabel, position[3], ifloat, jfloat, kfloat;
05302     double charge, hx, hy, hzed, *apos, mx, my, mz;
05303     int i, ii, jj, kk, nx, ny, nz, iatom;
05304     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
05305
05306
05307     VASSERT(thee != VNULL);
05308
05309     /* Get PBE info */
05310     pbe = thee->pbe;
05311     alist = pbe->alist;
05312     zmagic = Vpbe_getZmagic(pbe);
05313
05314     /* Mesh info */
05315     nx = thee->pmgp->nx;
05316     ny = thee->pmgp->ny;
05317     nz = thee->pmgp->nz;
05318     xlabel = thee->pmgp->hx;
05319     ylabel = thee->pmgp->hy;
05320     hzed = thee->pmgp->hzed;
05321
05322     /* Define the total domain size */
05323     xlabel = thee->pmgp->xlen;
05324     ylabel = thee->pmgp->ylen;
05325     zlabel = thee->pmgp->zlen;
05326
05327     /* Define the min/max dimensions */
05328     xmin = thee->pmgp->xcent - (xlabel/2.0);
05329     ymin = thee->pmgp->ycent - (ylabel/2.0);
05330     zmin = thee->pmgp->zcent - (zlabel/2.0);
05331     xmax = thee->pmgp->xcent + (xlabel/2.0);
05332     ymax = thee->pmgp->ycent + (ylabel/2.0);
05333     zmax = thee->pmgp->zcent + (zlabel/2.0);
05334
05335     /* Reset the charge array */
05336     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
```

```

05337
05338     /* Fill in the source term (atomic charges) */
05339     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05340     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05341
05342         atom = Valist_getAtom(alist, iatom);
05343         apos = Vatom_getPosition(atom);
05344         charge = Vatom_getCharge(atom);
05345
05346         /* Make sure we're on the grid */
05347         if ((apos[0]<=(xmin-hx)) || (apos[0]>=(xmax+hx)) || \
05348             (apos[1]<=(ymin-hy)) || (apos[1]>=(ymax+hy)) || \
05349             (apos[2]<=(zmin-hzed)) || (apos[2]>=(zmax+hzed))) {
05350             if ((thee->pmpg->bclf != BCFL_FOCUS) &&
05351                 (thee->pmpg->bclf != BCFL_MAP)) {
05352                 Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f, \
05353 %4.3f) is off the mesh (for cubic splines!!) (ignoring this atom):\n",
05354                     iatom, apos[0], apos[1], apos[2]);
05355                 Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
05356                     xmin, xmax);
05357                 Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
05358                     ymin, ymax);
05359                 Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
05360                     zmin, zmax);
05361             }
05362             fflush(stderr);
05363         } else {
05364
05365             /* Convert the atom position to grid reference frame */
05366             position[0] = apos[0] - xmin;
05367             position[1] = apos[1] - ymin;
05368             position[2] = apos[2] - zmin;
05369
05370             /* Scale the charge to be a delta function */
05371             charge = charge*zmagic/(hx*hy*hzed);
05372
05373             /* Figure out which vertices we're next to */
05374             ifloat = position[0]/hx;
05375             jfloat = position[1]/hy;
05376             kfloat = position[2]/hzed;
05377
05378             ip1    = (int)ceil(ifloat);
05379             ip2    = ip1 + 1;
05380             im1    = (int)floor(ifloat);
05381             im2    = im1 - 1;
05382             jp1    = (int)ceil(jfloat);
05383             jp2    = jp1 + 1;
05384             jm1    = (int)floor(jfloat);
05385             jm2    = jm1 - 1;
05386             kp1    = (int)ceil(kfloat);
05387             kp2    = kp1 + 1;
05388             km1    = (int)floor(kfloat);
05389             km2    = km1 - 1;
05390
05391             /* This step shouldn't be necessary, but it saves nasty debugging
05392              * later on if something goes wrong */
05393             ip2 = VMIN2(ip2,nx-1);

```

```

05394         ip1 = VMIN2(ip1,nx-1);
05395         im1 = VMAX2(im1,0);
05396         im2 = VMAX2(im2,0);
05397         jp2 = VMIN2(jp2,ny-1);
05398         jp1 = VMIN2(jp1,ny-1);
05399         jm1 = VMAX2(jm1,0);
05400         jm2 = VMAX2(jm2,0);
05401         kp2 = VMIN2(kp2,nz-1);
05402         kp1 = VMIN2(kp1,nz-1);
05403         km1 = VMAX2(km1,0);
05404         km2 = VMAX2(km2,0);
05405
05406         /* Now assign fractions of the charge to the nearby verts */
05407         for (ii=im2; ii<=ip2; ii++) {
05408             mx = bspline2(VFCHI(ii,ifloat));
05409             for (jj=jm2; jj<=jp2; jj++) {
05410                 my = bspline2(VFCHI(jj,jfloat));
05411                 for (kk=km2; kk<=kp2; kk++) {
05412                     mz = bspline2(VFCHI(kk,kfloat));
05413                     thee->charge[IJK(ii,jj,kk)] += (charge*mx*my*mz);
05414                 }
05415             }
05416         }
05417
05418     } /* endif (on the mesh) */
05419 } /* endfor (each atom) */
05420 }
05421
05422 VPUBLIC int Vpmg_fillco(Vpmg *thee,
05423     Vsurf_Meth surfMeth, double splineWin, Vchrg_Meth chargeMeth,
05424     int useDielXMap, Vgrid *dielXMap,
05425     int useDielyMap, Vgrid *dielyMap,
05426     int useDielZMap, Vgrid *dielZMap,
05427     int useKappaMap, Vgrid *kappaMap,
05428     int usePotMap, Vgrid *potMap,
05429     int useChargeMap, Vgrid *chargeMap) {
05430
05431     Vpbe *pbe;
05432     double xmin, xmax, ymin, ymax, zmin, zmax;
05433     double xlen, ylen, zlen, hx, hy, hzed;
05434     double epsw, epsp, ionstr;
05435     int i, nx, ny, nz, islap;
05436     Vrc_Codes rc;
05437
05438     if (thee == VNULL) {
05439         Vnm_print(2, "Vpmg_fillco: got NULL thee!\n");
05440         return 0;
05441     }
05442
05443     thee->surfMeth = surfMeth;
05444     thee->splineWin = splineWin;
05445     thee->chargeMeth = chargeMeth;
05446     thee->useDielXMap = useDielXMap;
05447     if (thee->useDielXMap) thee->dielXMap = dielXMap;
05448     thee->useDielyMap = useDielyMap;
05449     if (thee->useDielyMap) thee->dielyMap = dielyMap;
05450     thee->useDielZMap = useDielZMap;

```

```

05451     if (thee->useDielZMap) thee->dielZMap = dielZMap;
05452     thee->useKappaMap = useKappaMap;
05453     if (thee->useKappaMap) thee->kappaMap = kappaMap;
05454     thee->usePotMap = usePotMap;
05455     if (thee->usePotMap) thee->potMap = potMap;
05456     thee->useChargeMap = useChargeMap;
05457     if (thee->useChargeMap) thee->chargeMap = chargeMap;
05458
05459     /* Get PBE info */
05460     pbe = thee->pbe;
05461     ionstr = Vpbe_getBulkIonicStrength(pbe);
05462     epsw = Vpbe_getSolventDiel(pbe);
05463     epsp = Vpbe_getSoluteDiel(pbe);
05464
05465     /* Mesh info */
05466     nx = thee->pmgp->nx;
05467     ny = thee->pmgp->ny;
05468     nz = thee->pmgp->nz;
05469     hx = thee->pmgp->hx;
05470     hy = thee->pmgp->hy;
05471     hzed = thee->pmgp->hzed;
05472
05473     /* Define the total domain size */
05474     xlen = thee->pmgp->xlen;
05475     ylen = thee->pmgp->ylen;
05476     zlen = thee->pmgp->zlen;
05477
05478     /* Define the min/max dimensions */
05479     xmin = thee->pmgp->xcent - (xlen/2.0);
05480     thee->pmgp->xmin = xmin;
05481     ymin = thee->pmgp->ycent - (ylen/2.0);
05482     thee->pmgp->ymin = ymin;
05483     zmin = thee->pmgp->zcent - (zlen/2.0);
05484     thee->pmgp->zmin = zmin;
05485     xmax = thee->pmgp->xcent + (xlen/2.0);
05486     thee->pmgp->xmax = xmax;
05487     ymax = thee->pmgp->ycent + (ylen/2.0);
05488     thee->pmgp->ymax = ymax;
05489     zmax = thee->pmgp->zcent + (zlen/2.0);
05490     thee->pmgp->zmax = zmax;
05491     thee->rparm[2] = xmin;
05492     thee->rparm[3] = xmax;
05493     thee->rparm[4] = ymin;
05494     thee->rparm[5] = ymax;
05495     thee->rparm[6] = zmin;
05496     thee->rparm[7] = zmax;
05497
05498     /* This is a flag that gets set if the operator is a simple Laplacian;
05499      * i.e., in the case of a homogenous dielectric and zero ionic strength
05500      * The operator cannot be a simple Laplacian if maps are read in. */
05501     if(thee->useDielXMap || thee->useDielYMap || thee->useDielZMap ||
05502         thee->useKappaMap || thee->usePotMap){
05503         islap = 0;
05504     }else if ( (ionstr < VPMGSMALL) && (VABS(epsp-epsw) < VPMGSMALL) ){
05505         islap = 1;
05506     }else{
05507         islap = 0;

```

```

05508 }
05509
05510 /* Fill the mesh point coordinate arrays */
05511 for (i=0; i<nx; i++) thee->xf[i] = xmin + i*hx;
05512 for (i=0; i<ny; i++) thee->yf[i] = ymin + i*hy;
05513 for (i=0; i<nz; i++) thee->zf[i] = zmin + i*hzed;
05514
05515 /* Reset the tcf array */
05516 for (i=0; i<(nx*ny*nz); i++) thee->tcf[i] = 0.0;
05517
05518 /* Fill in the source term (atomic charges) */
05519 Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05520 rc = fillcoCharge(thee);
05521 switch(rc) {
05522 case VRC_SUCCESS:
05523 break;
05524 case VRC_WARNING:
05525 Vnm_print(2, "Vpmg_fillco: non-fatal errors while filling charge map!\n");
05526 break;
05527 case VRC_FAILURE:
05528 Vnm_print(2, "Vpmg_fillco: fatal errors while filling charge map!\n");
05529 return 0;
05530 break;
05531 }
05532
05533 /* THE FOLLOWING NEEDS TO BE DONE IF WE'RE NOT USING A SIMPLE LAPLACIAN
05534 * OPERATOR */
05535 if (!islap) {
05536 Vnm_print(0, "Vpmg_fillco: marking ion and solvent accessibility.\n");
05537 fillcoCoef(thee);
05538 Vnm_print(0, "Vpmg_fillco: done filling coefficient arrays\n");
05539
05540 } else /* else (!islap) ==> It's a Laplacian operator! */
05541
05542 for (i=0; i<(nx*ny*nz); i++) {
05543 thee->kappa[i] = 0.0;
05544 thee->epsx[i] = epsp;
05545 thee->epsy[i] = epsp;
05546 thee->epsz[i] = epsp;
05547 }
05548
05549 } /* endif (!islap) */
05550
05551 /* Fill the boundary arrays (except when focusing, bcfl = 4) */
05552 if (thee->pmpg->bcfl != BCFL_FOCUS) {
05553 Vnm_print(0, "Vpmg_fillco: filling boundary arrays\n");
05554 bcCalc(thee);
05555 Vnm_print(0, "Vpmg_fillco: done filling boundary arrays\n");
05556 }
05557
05558 thee->filled = 1;
05559
05560 return 1;
05561 }
05562
05563
05564 VPUBLIC int Vpmg_force(Vpmg *thee, double *force, int atomID,

```

```

05565     Vsurf_Meth srfm, Vchrg_Meth chgm) {
05566
05567     int rc = 1;
05568     double qff[3]; /* Charge-field force */
05569     double dbF[3]; /* Dielectric boundary force */
05570     double ibF[3]; /* Ion boundary force */
05571     double npF[3]; /* Non-polar boundary force */
05572
05573     VASSERT(thee != VNULL);
05574
05575     rc = rc && Vpmg_dbForce(thee, qff, atomID, srfm);
05576     rc = rc && Vpmg_ibForce(thee, dbF, atomID, srfm);
05577     rc = rc && Vpmg_qfForce(thee, ibF, atomID, chgm);
05578
05579     force[0] = qff[0] + dbF[0] + ibF[0];
05580     force[1] = qff[1] + dbF[1] + ibF[1];
05581     force[2] = qff[2] + dbF[2] + ibF[2];
05582
05583     return rc;
05584
05585 }
05586
05587 VPUBLIC int Vpmg_ibForce(Vpmg *thee, double *force, int atomID,
05588     Vsurf_Meth srfm) {
05589
05590     Valist *alist;
05591     Vacc *acc;
05592     Vpbe *pbe;
05593     Vatom *atom;
05594
05595     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
05596     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
05597     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
05598     double izmagic;
05599     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
05600
05601 /* For nonlinear forces */
05602     int ichop, nchop, nion, m;
05603     double ionConc[MAXION], ionRadii[MAXION], ionQ[MAXION], ionstr;
05604
05605     VASSERT(thee != VNULL);
05606
05607     acc = thee->pbe->acc;
05608     atom = Valist_getAtom(thee->pbe->alist, atomID);
05609     apos = Vatom_getPosition(atom);
05610     arad = Vatom_getRadius(atom);
05611
05612 /* Reset force */
05613     force[0] = 0.0;
05614     force[1] = 0.0;
05615     force[2] = 0.0;
05616
05617 /* Check surface definition */
05618     if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) && (srfm!=VSM_SPLINE4)) {
05619         Vnm_print(2, "Vpmg_ibForce: Forces *must* be calculated with \
05620 spline-based surfaces!\n");
05621         Vnm_print(2, "Vpmg_ibForce: Skipping ionic boundary force \

```

```

05622 calculation!\n");
05623     return 0;
05624 }
05625
05626 /* If we aren't in the current position, then we're done */
05627 if (atom->partID == 0) return 1;
05628
05629 /* Get PBE info */
05630 pbe = theee->pbe;
05631 acc = pbe->acc;
05632 alist = pbe->alist;
05633 irad = Vpbe_getMaxIonRadius(pbe);
05634 zkappa2 = Vpbe_getZkappa2(pbe);
05635 izmagic = 1.0/Vpbe_getZmagic(pbe);
05636
05637 ionstr = Vpbe_getBulkIonicStrength(pbe);
05638 Vpbe_getIons(pbe, &nion, ionConc, ionRadii, ionQ);
05639
05640 /* Mesh info */
05641 nx = theee->pmgp->nx;
05642 ny = theee->pmgp->ny;
05643 nz = theee->pmgp->nz;
05644 hx = theee->pmgp->hx;
05645 hy = theee->pmgp->hy;
05646 hzed = theee->pmgp->hzed;
05647 xlen = theee->pmgp->xlen;
05648 ylen = theee->pmgp->ylen;
05649 zlen = theee->pmgp->zlen;
05650 xmin = theee->pmgp->xmin;
05651 ymin = theee->pmgp->ymin;
05652 zmin = theee->pmgp->zmin;
05653 xmax = theee->pmgp->xmax;
05654 ymax = theee->pmgp->ymax;
05655 zmax = theee->pmgp->zmax;
05656
05657 /* Sanity check: there is no force if there is zero ionic strength */
05658 if (zkappa2 < VPMGSMAL) {
05659 #ifndef VAPBSQUIET
05660     Vnm_print(2, "Vpmg_ibForce: No force for zero ionic strength!\n");
05661 #endif
05662     return 1;
05663 }
05664
05665 /* Make sure we're on the grid */
05666 if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
05667     (apos[1]<=ymin) || (apos[1]>=ymax) || \
05668     (apos[2]<=zmin) || (apos[2]>=zmax)) {
05669 if ((theee->pmgp->bcfl != BCFL_FOCUS) &&
05670 (thee->pmgp->bcfl != BCFL_MAP)) {
05671     Vnm_print(2, "Vpmg_ibForce: Atom #d at (%4.3f, %4.3f, %4.3f) is off
the mesh (ignoring):\n",
05672             atom, apos[0], apos[1], apos[2]);
05673     Vnm_print(2, "Vpmg_ibForce: xmin = %g, xmax = %g\n",
05674             xmin, xmax);
05675     Vnm_print(2, "Vpmg_ibForce: ymin = %g, ymax = %g\n",
05676             ymin, ymax);
05677     Vnm_print(2, "Vpmg_ibForce: zmin = %g, zmax = %g\n",

```

```

05678         zmin, zmax);
05679     }
05680     fflush(stderr);
05681 } else {
05682
05683     /* Convert the atom position to grid reference frame */
05684     position[0] = apos[0] - xmin;
05685     position[1] = apos[1] - ymin;
05686     position[2] = apos[2] - zmin;
05687
05688     /* Integrate over points within this atom's (inflated) radius */
05689     rtot = (irad + arad + thee->splineWin);
05690     rtot2 = VSQR(rtot);
05691     dx = rtot + 0.5*hx;
05692     imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
05693     imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
05694     for (i=imin; i<=imax; i++) {
05695         dx2 = VSQR(position[0] - hx*i);
05696         if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
05697         else dy = 0.5*hy;
05698         jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
05699         jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
05700         for (j=jmin; j<=jmax; j++) {
05701             dy2 = VSQR(position[1] - hy*j);
05702             if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
05703             else dz = 0.5*hzed;
05704             kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
05705             kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
05706             for (k=kmin; k<=kmax; k++) {
05707                 dz2 = VSQR(k*hzed - position[2]);
05708                 /* See if grid point is inside ivdw radius and set kappa
05709                  * accordingly (do spline assignment here) */
05710                 if ((dz2 + dy2 + dx2) <= rtot2) {
05711                     gpos[0] = i*hx + xmin;
05712                     gpos[1] = j*hy + ymin;
05713                     gpos[2] = k*hzed + zmin;
05714
05715                 /* Select the correct function based on the surface definition
05716                  * (now including the 7th order polynomial) */
05717                 Vpmg_splineSelect(srfm,acc, gpos,thee->splineWin, irad, atom, tgrad);
05718
05719                 if (thee->pmgp->nonlin) {
05720                     /* Nonlinear forces */
05721                     fmag = 0.0;
05722                     nchop = 0;
05723                     for (m=0; m<nion; m++) {
05724                         fmag += (thee->kappa[IJK(i,j,k)])*ionConc[m]*(
05725                         Vcap_exp(-ionQ[m]*thee->u[IJK(i,j,k)], &ichop)-1.0)/ionstr;
05726                         nchop += ichop;
05727                     }
05728                     /* if (nchop > 0) Vnm_print(2, "Vpmg_ibForce: Chopped EXP %d ti
05729                     mes!\n", nchop); */
05730                     force[0] += (zkappa2*fmag*tgrad[0]);
05731                     force[1] += (zkappa2*fmag*tgrad[1]);
05732                     force[2] += (zkappa2*fmag*tgrad[2]);
05733                 } else {
05734                     /* Use of bulk factor (zkappa2) OK here because

```

```

05733             * LPBE force approximation */
05734             /* NAB -- did we forget a kappa factor here??? */
05735             fmag = VSQR(thee->u[IJK(i,j,k)])*(thee->kappa[IJK(i,j
05736 ,k)]);
05737             force[0] += (zkappa2*fmag*tgrad[0]);
05738             force[1] += (zkappa2*fmag*tgrad[1]);
05739             force[2] += (zkappa2*fmag*tgrad[2]);
05740         }
05741     } /* k loop */
05742   } /* j loop */
05743 } /* i loop */
05744 }
05745 force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
05746 force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
05747 force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
05748
05749 return 1;
05750 }
05751
05752 VPUBLIC int Vpmg_dbForce(Vpmg *thee, double *dbForce, int atomID,
05753   Vsurf_Meth srfm) {
05754
05755   Vacc *acc;
05756   Vpbe *pbe;
05757   Vatom *atom;
05758
05759   double *apos, position[3], arad, srad, hx, hy, hzed, izmagic, deps,depsi;
05760   double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
05761   double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
05762   double *u, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijmk, Hzijkml;
05763   double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijmk[3];
05764   double dHzijkml[3];
05765   int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
05766
05767   VASSERT(thee != VNNULL);
05768   if (!thee->filled) {
05769     Vnm_print(2, "Vpmg_dbForce: Need to callVpmg_fillco!\n");
05770     return 0;
05771   }
05772
05773   acc = thee->pbe->acc;
05774   atom = Valist_getAtom(thee->pbe->alist, atomID);
05775   apos = Vatom_getPosition(atom);
05776   arad = Vatom_getRadius(atom);
05777   srad = Vpbe_getSolventRadius(thee->pbe);
05778
05779 /* Reset force */
05780   dbForce[0] = 0.0;
05781   dbForce[1] = 0.0;
05782   dbForce[2] = 0.0;
05783
05784 /* Check surface definition */
05785   if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) && (srfm!=VSM_SPLINE4)) {
05786     Vnm_print(2, "Vpmg_dbForce: Forces *must* be calculated with \
05787 spline-based surfaces!\n");
05788     Vnm_print(2, "Vpmg_dbForce: Skipping dielectric/apolar boundary \

```

```

05789 force calculation!\n");
05790     return 0;
05791 }
05792
05793
05794 /* If we aren't in the current position, then we're done */
05795 if (atom->partID == 0) return 1;
05796
05797 /* Get PBE info */
05798 pbe = theee->pbe;
05799 acc = pbe->acc;
05800 epsp = Vpbe_getSoluteDiel(pbe);
05801 epsw = Vpbe_getSolventDiel(pbe);
05802 kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
05803 izmagic = 1.0/Vpbe_getZmagic(pbe);
05804
05805 /* Mesh info */
05806 nx = theee->pmgp->nx;
05807 ny = theee->pmgp->ny;
05808 nz = theee->pmgp->nz;
05809 hx = theee->pmgp->hx;
05810 hy = theee->pmgp->hy;
05811 hzed = theee->pmgp->hzed;
05812 xlen = theee->pmgp->xlen;
05813 ylen = theee->pmgp->ylen;
05814 zlen = theee->pmgp->zlen;
05815 xmin = theee->pmgp->xmin;
05816 ymin = theee->pmgp->ymin;
05817 zmin = theee->pmgp->zmin;
05818 xmax = theee->pmgp->xmax;
05819 ymax = theee->pmgp->ymax;
05820 zmax = theee->pmgp->zmax;
05821 u = theee->u;
05822
05823 /* Sanity check: there is no force if there is zero ionic strength */
05824 if (VABS(epsp-epsw) < VPMGSMALL) {
05825 Vnm_print(0, "Vpmg_dbForce: No force for uniform dielectric!\n");
05826 return 1;
05827 }
05828 deps = (epsw - epsp);
05829 depsi = 1.0/deps;
05830 rtot = (arad + theee->splineWin + srad);
05831
05832 /* Make sure we're on the grid */
05833 /* Grid checking modified by Matteo Rotter */
05834 if ((apos[0]<=xmin + rtot) || (apos[0]>=xmax - rtot) || \
05835 (apos[1]<=ymin + rtot) || (apos[1]>=ymax - rtot) || \
05836 (apos[2]<=zmin + rtot) || (apos[2]>=zmax - rtot)) {
05837 if ((theee->pmgp->bcfl != BCFL_FOCUS) &&
05838 (theee->pmgp->bcfl != BCFL_MAP)) {
05839     Vnm_print(2, "Vpmg_dbForce: Atom #%d at (%4.3f, %4.3f, %4.3f) is off
the mesh (ignoring):\n",
05840     atomID, apos[0], apos[1], apos[2]);
05841     Vnm_print(2, "Vpmg_dbForce: xmin = %g, xmax = %g\n",
05842     xmin, xmax);
05843     Vnm_print(2, "Vpmg_dbForce: ymin = %g, ymax = %g\n",
05844     ymin, ymax);

```

```

05845         Vnm_print(2, "Vpmg_dbForce:      zmin = %g, zmax = %g\n",
05846             zmin, zmax);
05847         }
05848         fflush(stderr);
05849     } else {
05850
05851     /* Convert the atom position to grid reference frame */
05852     position[0] = apos[0] - xmin;
05853     position[1] = apos[1] - ymin;
05854     position[2] = apos[2] - zmin;
05855
05856     /* Integrate over points within this atom's (inflated) radius */
05857     rtot2 = VSQR(rtot);
05858     dx = rtot/hx;
05859     imin = (int)floor((position[0]-rtot)/hx);
05860     if (imin < 1) {
05861         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
05862         return 0;
05863     }
05864     imax = (int)ceil((position[0]+rtot)/hx);
05865     if (imax > (nx-2)) {
05866         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
05867         return 0;
05868     }
05869     jmin = (int)floor((position[1]-rtot)/hy);
05870     if (jmin < 1) {
05871         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
05872         return 0;
05873     }
05874     jmax = (int)ceil((position[1]+rtot)/hy);
05875     if (jmax > (ny-2)) {
05876         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
05877         return 0;
05878     }
05879     kmin = (int)floor((position[2]-rtot)/hzed);
05880     if (kmin < 1) {
05881         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
05882         return 0;
05883     }
05884     kmax = (int)ceil((position[2]+rtot)/hzed);
05885     if (kmax > (nz-2)) {
05886         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
05887         return 0;
05888     }
05889     for (i=imin; i<=imax; i++) {
05890         for (j=jmin; j<=jmax; j++) {
05891             for (k=kmin; k<=kmax; k++) {
05892                 /* i,j,k */
05893                 gpos[0] = (i+0.5)*hx + xmin;
05894                 gpos[1] = j*hy + ymin;
05895                 gpos[2] = k*hzed + zmin;
05896                 Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
05897
05898             /* Select the correct function based on the surface definition
05899             * (now including the 7th order polynomial) */
05900             Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHxijk);
05901         /*

```

```

05902     switch (srfm) {
05903         case VSM_SPLINE :
05904             Vacc_splineAccGradAtomNorm(acc, gpos, thee->splineWin, 0.,
05905                         atom, dHxijk);
05906             break;
05907         case VSM_SPLINE4 :
05908             Vacc_splineAccGradAtomNorm4(acc, gpos, thee->splineWin, 0.,
05909                         atom, dHxijk);
05910             break;
05911         default:
05912             Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
05913             return;
05914     }
05915 */
05916     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
05917     gpos[0] = i*hx + xmin;
05918     gpos[1] = (j+0.5)*hy + ymin;
05919     gpos[2] = k*hzed + zmin;
05920     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
05921
05922 /* Select the correct function based on the surface definition
05923 * (now including the 7th order polynomial) */
05924 Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHyijk);
05925
05926 for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
05927     gpos[0] = i*hx + xmin;
05928     gpos[1] = j*hy + ymin;
05929     gpos[2] = (k+0.5)*hzed + zmin;
05930     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
05931
05932 /* Select the correct function based on the surface definition
05933 * (now including the 7th order polynomial) */
05934 Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHzijk);
05935
05936 for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
05937 /* i-1,j,k */
05938     gpos[0] = (i-0.5)*hx + xmin;
05939     gpos[1] = j*hy + ymin;
05940     gpos[2] = k*hzed + zmin;
05941     Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
05942
05943 /* Select the correct function based on the surface definition
05944 * (now including the 7th order polynomial) */
05945 Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHxim1jk);
05946
05947 for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
05948 /* i,j-1,k */
05949     gpos[0] = i*hx + xmin;
05950     gpos[1] = (j-0.5)*hy + ymin;
05951     gpos[2] = k*hzed + zmin;
05952     Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
05953
05954 /* Select the correct function based on the surface definition
05955 * (now including the 7th order polynomial) */
05956 Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHyijm1k);
05957
05958 for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;

```

```

05959             /* i,j,k-1 */
05960             gpos[0] = i*hx + xmin;
05961             gpos[1] = j*hy + ymin;
05962             gpos[2] = (k-0.5)*hzed + zmin;
05963             Hzijkml = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
05964
05965             /* Select the correct function based on the surface definition
05966             * (now including the 7th order polynomial) */
05967             Vpmg_splineSelect(srfm,acc,gpos,thee->splineWin,0.,atom,dHzijkml);
05968
05969             for (l=0; l<3; l++) dHzijkml[l] *= Hzijkml;
05970                     /* *** CALCULATE DIELECTRIC BOUNDARY FORCES *** */
05971                     dbFmag = u[IJK(i,j,k)];
05972                     tgrad[0] =
05973                     (dHxijk[0] *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
05974                     + dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
05975                     + (dHyijk[0] *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
05976                     + dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
05977                     + (dHzijk[0] *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
05978                     + dHzijkml[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
05979                     tgrad[1] =
05980                     (dHxijk[1] *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
05981                     + dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
05982                     + (dHyijk[1] *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
05983                     + dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
05984                     + (dHzijk[1] *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
05985                     + dHzijkml[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
05986                     tgrad[2] =
05987                     (dHxijk[2] *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
05988                     + dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
05989                     + (dHyijk[2] *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
05990                     + dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
05991                     + (dHzijk[2] *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
05992                     + dHzijkml[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
05993             dbForce[0] += (dbFmag*tgrad[0]);
05994             dbForce[1] += (dbFmag*tgrad[1]);
05995             dbForce[2] += (dbFmag*tgrad[2]);
05996
05997         } /* k loop */
05998     } /* j loop */
05999 } /* i loop */
06000
06001             dbForce[0] = -dbForce[0]*hx*hy*hzed*deps*0.5*izmagic;
06002             dbForce[1] = -dbForce[1]*hx*hy*hzed*deps*0.5*izmagic;
06003             dbForce[2] = -dbForce[2]*hx*hy*hzed*deps*0.5*izmagic;
06004 }
06005
06006     return 1;
06007 }
06008
06009 VPUBLIC int Vpmg_qfForce(Vpmg *thee, double *force, int atomID,
06010     Vchrg_Meth chgm) {
06011
06012     double tforce[3];
06013
06014     /* Reset force */
06015     force[0] = 0.0;

```

```

06016     force[1] = 0.0;
06017     force[2] = 0.0;
06018
06019     /* Check surface definition */
06020     if (chgm != VCM_BSPL2) {
06021         Vnm_print(2, "Vpmg_qfForce: It is recommended that forces be \
06022 calculated with the\n");
06023         Vnm_print(2, "Vpmg_qfForce: cubic spline charge discretization \
06024 scheme\n");
06025     }
06026
06027     switch (chgm) {
06028         case VCM_TRIL:
06029             qfForceSpline1(thee, tforce, atomID);
06030             break;
06031         case VCM_BSPL2:
06032             qfForceSpline2(thee, tforce, atomID);
06033             break;
06034     case VCM_BSPL4:
06035         qfForceSpline4(thee, tforce, atomID);
06036         break;
06037         default:
06038             Vnm_print(2, "Vpmg_qfForce: Undefined charge discretization \
06039 method (%d)\n", chgm);
06040             Vnm_print(2, "Vpmg_qfForce: Forces not calculated!\n");
06041             return 0;
06042     }
06043
06044     /* Assign forces */
06045     force[0] = tforce[0];
06046     force[1] = tforce[1];
06047     force[2] = tforce[2];
06048
06049     return 1;
06050 }
06051
06052
06053 VPRIIVATE void qfForceSpline1(Vpmg *thee, double *force, int atomID) {
06054
06055     Vatom *atom;
06056
06057     double *apos, position[3], hx, hy, hzed;
06058     double xmin, ymin, zmin, xmax, ymax, zmax;
06059     double dx, dy, dz;
06060     double *u, charge, ifloat, jfloat, kfloat;
06061     int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
06062
06063     VASSERT(thee != VNNULL);
06064
06065     atom = Valist_getAtom(thee->pbe->alist, atomID);
06066     apos = VatomGetPosition(atom);
06067     charge = Vatom_getCharge(atom);
06068
06069     /* Reset force */
06070     force[0] = 0.0;
06071     force[1] = 0.0;
06072     force[2] = 0.0;

```

```

06073
06074     /* If we aren't in the current position, then we're done */
06075     if (atom->partID == 0) return;
06076
06077     /* Mesh info */
06078     nx = thee->pmgp->nx;
06079     ny = thee->pmgp->ny;
06080     nz = thee->pmgp->nz;
06081     hx = thee->pmgp->hx;
06082     hy = thee->pmgp->hy;
06083     hzed = thee->pmgp->hzed;
06084     xmin = thee->pmgp->xmin;
06085     ymin = thee->pmgp->ymin;
06086     zmin = thee->pmgp->zmin;
06087     xmax = thee->pmgp->xmax;
06088     ymax = thee->pmgp->ymax;
06089     zmax = thee->pmgp->zmax;
06090     u = thee->u;
06091
06092     /* Make sure we're on the grid */
06093     if ((apos[0]<=xmin) || (apos[0]>=xmax) || (apos[1]<=ymin) || \
06094         (apos[1]>=ymax) || (apos[2]<=zmin) || (apos[2]>=zmax)) {
06095     if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06096         (thee->pmgp->bcfl != BCFL_MAP)) {
06097         Vnm_print(2, "Vpmg_qfForce: Atom #d at (%4.3f, %4.3f, %4.3f) is off
the mesh (ignoring):\n", atomID, apos[0], apos[1], apos[2]);
06098         Vnm_print(2, "Vpmg_qfForce: xmin = %g, xmax = %g\n", xmin, xmax);
06099         Vnm_print(2, "Vpmg_qfForce: ymin = %g, ymax = %g\n", ymin, ymax);
06100         Vnm_print(2, "Vpmg_qfForce: zmin = %g, zmax = %g\n", zmin, zmax);
06101     }
06102     fflush(stderr);
06103 } else {
06104
06105     /* Convert the atom position to grid coordinates */
06106     position[0] = apos[0] - xmin;
06107     position[1] = apos[1] - ymin;
06108     position[2] = apos[2] - zmin;
06109     ifloat = position[0]/hx;
06110     jfloat = position[1]/hy;
06111     kfloat = position[2]/hzed;
06112     ihi = (int)ceil(ifloat);
06113     ilo = (int)floor(ifloat);
06114     jhi = (int)ceil(jfloat);
06115     jlo = (int)floor(jfloat);
06116     khi = (int)ceil(kfloat);
06117     klo = (int)floor(kfloat);
06118     VASSERT((ihi < nx) && (ihi >=0));
06119     VASSERT((ilo < nx) && (ilo >=0));
06120     VASSERT((jhi < ny) && (jhi >=0));
06121     VASSERT((jlo < ny) && (jlo >=0));
06122     VASSERT((khi < nz) && (khi >=0));
06123     VASSERT((klo < nz) && (klo >=0));
06124     dx = ifloat - (double)(ilo);
06125     dy = jfloat - (double)(jlo);
06126     dz = kfloat - (double)(klo);
06127
06128

```

```

06129 #if 0
06130     Vnm_print(1, "Vpmg_qfForce: (DEBUG) u ~ %g\n",
06131             dx      *dy      *dz      *u[IJK(ihi,jhi,khi)]
06132             +dx      *dy      *(1-dz)*u[IJK(ihi,jhi,klo)]
06133             +dx      *(1-dy)*dz      *u[IJK(ihi,jlo,khi)]
06134             +dx      *(1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
06135             +(1-dx)*dy      *dz      *u[IJK(ihi,jhi,khi)]
06136             +(1-dx)*dy      *(1-dz)*u[IJK(ihi,jhi,klo)]
06137             +(1-dx)*(1-dy)*dz      *u[IJK(ihi,jlo,khi)]
06138             +(1-dx)*(1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]);
06139 #endif
06140
06141
06142     if ((dx > VPMGSMALL) && (VABS(1.0-dx) > VPMGSMALL)) {
06143         force[0] =
06144             -charge*(dy      *dz      *u[IJK(ihi,jhi,khi)]
06145                     + dy      *(1-dz)*u[IJK(ihi,jhi,klo)]
06146                     + (1-dy)*dz      *u[IJK(ihi,jlo,khi)]
06147                     + (1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
06148                     - dy      *dz      *u[IJK(ihi,jhi,khi)]
06149                     - dy      *(1-dz)*u[IJK(ihi,jhi,klo)]
06150                     - (1-dy)*dz      *u[IJK(ihi,jlo,khi)]
06151                     - (1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)])/hx;
06152     } else {
06153         force[0] = 0;
06154         Vnm_print(0,
06155                 "Vpmg_qfForce: Atom %d on x gridline; zero x-force\n", atomID);
06156     }
06157     if ((dy > VPMGSMALL) && (VABS(1.0-dy) > VPMGSMALL)) {
06158         force[1] =
06159             -charge*(dx      *dz      *u[IJK(ihi,jhi,khi)]
06160                     + dx      *(1-dz)*u[IJK(ihi,jhi,klo)]
06161                     - dx      *dz      *u[IJK(ihi,jlo,khi)]
06162                     - dx      *(1-dz)*u[IJK(ihi,jlo,klo)]
06163                     + (1-dx)*dz      *u[IJK(ihi,jhi,khi)]
06164                     + (1-dx)*(1-dz)*u[IJK(ihi,jhi,klo)]
06165                     - (1-dx)*dz      *u[IJK(ihi,jlo,khi)]
06166                     - (1-dx)*(1-dz)*u[IJK(ihi,jlo,klo)])/hy;
06167     } else {
06168         force[1] = 0;
06169         Vnm_print(0,
06170                 "Vpmg_qfForce: Atom %d on y gridline; zero y-force\n", atomID);
06171     }
06172     if ((dz > VPMGSMALL) && (VABS(1.0-dz) > VPMGSMALL)) {
06173         force[2] =
06174             -charge*(dy      *dx      *u[IJK(ihi,jhi,khi)]
06175                     - dy      *dx      *u[IJK(ihi,jhi,klo)]
06176                     + (1-dy)*dx      *u[IJK(ihi,jlo,khi)]
06177                     - (1-dy)*dx      *u[IJK(ihi,jlo,klo)]
06178                     + dy      *(1-dx)*u[IJK(ihi,jhi,khi)]
06179                     - dy      *(1-dx)*u[IJK(ihi,jhi,klo)]
06180                     + (1-dy)*(1-dx)*u[IJK(ihi,jlo,khi)]
06181                     - (1-dy)*(1-dx)*u[IJK(ihi,jlo,klo)])/hzed;
06182     } else {
06183         force[2] = 0;
06184         Vnm_print(0,
06185                 "Vpmg_qfForce: Atom %d on z gridline; zero z-force\n", atomID);

```

```

06186         }
06187     }
06188 }
06189
06190 VPRIVATE void qfForceSpline2(Vpmg *thee, double *force, int atomID) {
06191
06192     Vatom *atom;
06193
06194     double *apos, position[3], hx, hy, hzed;
06195     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
06196     double mx, my, mz, dmx, dmy, dmz;
06197     double *u, charge, ifloat, jfloat, kfloat;
06198     int nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
06199     int kp1, kp2, ii, jj, kk;
06200
06201     VASSERT(thee != VNULL);
06202
06203     atom = Valist_getAtom(thee->pbe->alist, atomID);
06204     apos = VatomGetPosition(atom);
06205     charge = Vatom_getCharge(atom);
06206
06207     /* Reset force */
06208     force[0] = 0.0;
06209     force[1] = 0.0;
06210     force[2] = 0.0;
06211
06212     /* If we aren't in the current position, then we're done */
06213     if (atom->partID == 0) return;
06214
06215     /* Mesh info */
06216     nx = thee->pmgp->nx;
06217     ny = thee->pmgp->ny;
06218     nz = thee->pmgp->nz;
06219     hx = thee->pmgp->hx;
06220     hy = thee->pmgp->hy;
06221     hzed = thee->pmgp->hzed;
06222     xlen = thee->pmgp->xlen;
06223     ylen = thee->pmgp->ylen;
06224     zlen = thee->pmgp->zlen;
06225     xmin = thee->pmgp->xmin;
06226     ymin = thee->pmgp->ymin;
06227     zmin = thee->pmgp->zmin;
06228     xmax = thee->pmgp->xmax;
06229     ymax = thee->pmgp->ymax;
06230     zmax = thee->pmgp->zmax;
06231     u = thee->u;
06232
06233     /* Make sure we're on the grid */
06234     if ((apos[0]<=(xmin+hx)) || (apos[0]>=(xmax-hx)) \
06235         || (apos[1]<=(ymin+hy)) || (apos[1]>=(ymax-hy)) \
06236         || (apos[2]<=(zmin+hzed)) || (apos[2]>=(zmax-hzed))) {
06237         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06238             (thee->pmgp->bcfl != BCFL_MAP)) {
06239             Vnm_print(2, "qfForceSpline2: Atom #%-d off the mesh \
06240             (ignoring)\n", atomID);
06241             }
06242             fflush(stderr);

```

```

06243
06244     } else {
06245
06246         /* Convert the atom position to grid coordinates */
06247         position[0] = apos[0] - xmin;
06248         position[1] = apos[1] - ymin;
06249         position[2] = apos[2] - zmin;
06250         ifloat = position[0]/hx;
06251         jfloat = position[1]/hy;
06252         kfloat = position[2]/hzed;
06253         ip1 = (int)ceil(ifloat);
06254         ip2 = ip1 + 1;
06255         im1 = (int)floor(ifloat);
06256         im2 = im1 - 1;
06257         jp1 = (int)ceil(jfloat);
06258         jp2 = jp1 + 1;
06259         jm1 = (int)floor(jfloat);
06260         jm2 = jm1 - 1;
06261         kp1 = (int)ceil(kfloat);
06262         kp2 = kp1 + 1;
06263         km1 = (int)floor(kfloat);
06264         km2 = km1 - 1;
06265
06266         /* This step shouldn't be necessary, but it saves nasty debugging
06267         * later on if something goes wrong */
06268         ip2 = VMIN2(ip2,nx-1);
06269         ip1 = VMIN2(ip1,nx-1);
06270         im1 = VMAX2(im1,0);
06271         im2 = VMAX2(im2,0);
06272         jp2 = VMIN2(jp2,ny-1);
06273         jp1 = VMIN2(jp1,ny-1);
06274         jm1 = VMAX2(jm1,0);
06275         jm2 = VMAX2(jm2,0);
06276         kp2 = VMIN2(kp2,nz-1);
06277         kp1 = VMIN2(kp1,nz-1);
06278         km1 = VMAX2(km1,0);
06279         km2 = VMAX2(km2,0);
06280
06281
06282     for (ii=im2; ii<=ip2; ii++) {
06283         mx = bspline2(VFCHI(ii,ifloat));
06284         dmx = dbspline2(VFCHI(ii,ifloat));
06285         for (jj=jm2; jj<=jp2; jj++) {
06286             my = bspline2(VFCHI(jj,jfloat));
06287             dmy = dbspline2(VFCHI(jj,jfloat));
06288             for (kk=km2; kk<=kp2; kk++) {
06289                 mz = bspline2(VFCHI(kk,kfloat));
06290                 dmz = dbspline2(VFCHI(kk,kfloat));
06291
06292                 force[0] += (charge*dmx*my*mz*u[IJK(ii,jj,kk)])/hx;
06293                 force[1] += (charge*mx*dmy*mz*u[IJK(ii,jj,kk)])/hy;
06294                 force[2] += (charge*mx*my*dmz*u[IJK(ii,jj,kk)])/hzed;
06295             }
06296         }
06297     }
06298 }
06299

```

```

06300     }
06301 }
06302
06303 VPRIVATE void qfForceSpline4(Vpmg *thee, double *force, int atomID) {
06304
06305     Vatom *atom;
06306     double f, c, *u, *apos, position[3];
06307
06308     /* Grid variables */
06309     int nx, ny, nz;
06310     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
06311     double hx, hy, hzed, ifloat, jfloat, kfloat;
06312
06313     /* B-spline weights */
06314     double mx, my, mz, dmx, dmy, dmz;
06315     double mi, mj, mk;
06316
06317     /* Loop indeces */
06318     int i, j, k, ii, jj, kk;
06319     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
06320
06321     /* field */
06322     double e[3];
06323
06324     VASSERT(thee != VNULL);
06325     VASSERT(thee->filled);
06326
06327     atom = Valist_getAtom(thee->pbe->alist, atomID);
06328     apos = VatomGetPosition(atom);
06329     c = Vatom_getCharge(atom);
06330
06331     for (i=0;i<3;i++){
06332         e[i] = 0.0;
06333     }
06334
06335     /* Mesh info */
06336     nx = thee->pmgp->nx;
06337     ny = thee->pmgp->ny;
06338     nz = thee->pmgp->nz;
06339     hx = thee->pmgp->hx;
06340     hy = thee->pmgp->hy;
06341     hzed = thee->pmgp->hzed;
06342     xlen = thee->pmgp->xlen;
06343     ylen = thee->pmgp->ylen;
06344     zlen = thee->pmgp->zlen;
06345     xmin = thee->pmgp->xmin;
06346     ymin = thee->pmgp->ymin;
06347     zmin = thee->pmgp->zmin;
06348     xmax = thee->pmgp->xmax;
06349     ymax = thee->pmgp->ymax;
06350     zmax = thee->pmgp->zmax;
06351     u = thee->u;
06352
06353     /* Make sure we're on the grid */
06354     if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
06355     || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
06356     || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {

```

```

06357     Vnm_print(2, "qffForceSpline4: Atom off the mesh \
06358     (ignoring) %6.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
06359         fflush(stderr);
06360     } else {
06361         /* Convert the atom position to grid coordinates */
06362         position[0] = apos[0] - xmin;
06363         position[1] = apos[1] - ymin;
06364         position[2] = apos[2] - zmin;
06365         ifloat = position[0]/hx;
06366         jfloat = position[1]/hy;
06367         kfloat = position[2]/hzed;
06368         ip1 = (int)ceil(ifloat);
06369         ip2 = ip1 + 2;
06370         im1 = (int)floor(ifloat);
06371         im2 = im1 - 2;
06372         jp1 = (int)ceil(jfloat);
06373         jp2 = jp1 + 2;
06374         jm1 = (int)floor(jfloat);
06375         jm2 = jm1 - 2;
06376         kp1 = (int)ceil(kfloat);
06377         kp2 = kp1 + 2;
06378         km1 = (int)floor(kfloat);
06379         km2 = km1 - 2;
06380
06381         /* This step shouldn't be necessary, but it saves nasty debugging
06382 * later on if something goes wrong */
06383         ip2 = VMIN2(ip2,nx-1);
06384         ip1 = VMIN2(ip1,nx-1);
06385         im1 = VMAX2(im1,0);
06386         im2 = VMAX2(im2,0);
06387         jp2 = VMIN2(jp2,ny-1);
06388         jp1 = VMIN2(jp1,ny-1);
06389         jm1 = VMAX2(jm1,0);
06390         jm2 = VMAX2(jm2,0);
06391         kp2 = VMIN2(kp2,nz-1);
06392         kp1 = VMIN2(kp1,nz-1);
06393         km1 = VMAX2(km1,0);
06394         km2 = VMAX2(km2,0);
06395
06396         for (ii=im2; ii<ip2; ii++) {
06397             mi = VFCHI4(ii,ifloat);
06398             mx = bspline4(mi);
06399             dmx = dbspline4(mi);
06400             for (jj=jm2; jj<=jp2; jj++) {
06401                 mj = VFCHI4(jj,jfloat);
06402                 my = bspline4(mj);
06403                 dmy = dbspline4(mj);
06404                 for (kk=km2; kk<=kp2; kk++) {
06405                     mk = VFCHI4(kk,kfloat);
06406                     mz = bspline4(mk);
06407                     dmz = dbspline4(mk);
06408                     f = u[IJK(ii,jj,kk)];
06409                     /* Field */
06410                     e[0] += f*dmx*my*mz/hx;
06411                     e[1] += f*mx*dmy*mz/hy;
06412                     e[2] += f*mx*my*dmz/hzed;
06413

```

```

06414         }
06415     }
06416   }
06417 }
06418
06419 /* Monopole Force */
06420 force[0] = e[0]*c;
06421 force[1] = e[1]*c;
06422 force[2] = e[2]*c;
06423
06424 }
06425
06426 VPRIIVATE void markFrac(
06427     double rtot, double *tpos,
06428     int nx, int ny, int nz,
06429     double hx, double hy, double hzed,
06430     double xmin, double ymin, double zmin,
06431     double *xarray, double *yarray, double *zarray) {
06432
06433     int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
06434     double dx, dx2, dy, dy2, dz, dz2, a000, a001, a010, a100, r2;
06435     double x, xp, xm, y, yp, ym, zp, z, zm, xspan, yspan, zspan;
06436     double rtot2, pos[3];
06437
06438 /* Convert to grid reference frame */
06439 pos[0] = tpos[0] - xmin;
06440 pos[1] = tpos[1] - ymin;
06441 pos[2] = tpos[2] - zmin;
06442
06443     rtot2 = VSQR(rtot);
06444
06445     xspan = rtot + 2*hx;
06446     imin = VMAX2(0, (int)ceil((pos[0] - xspan)/hx));
06447     imax = VMIN2(nx-1, (int)floor((pos[0] + xspan)/hx));
06448     for (i=imin; i<=imax; i++) {
06449         x = hx*i;
06450         dx2 = VSQR(pos[0] - x);
06451         if (rtot2 > dx2) {
06452             yspan = VSQRT(rtot2 - dx2) + 2*hy;
06453         } else {
06454             yspan = 2*hy;
06455         }
06456         jmin = VMAX2(0, (int)ceil((pos[1] - yspan)/hy));
06457         jmax = VMIN2(ny-1, (int)floor((pos[1] + yspan)/hy));
06458         for (j=jmin; j<=jmax; j++) {
06459             y = hy*j;
06460             dy2 = VSQR(pos[1] - y);
06461             if (rtot2 > (dx2+dy2)) {
06462                 zspan = VSQRT(rtot2-dx2-dy2) + 2*hzed;
06463             } else {
06464                 zspan = 2*hzed;
06465             }
06466             kmin = VMAX2(0, (int)ceil((pos[2] - zspan)/hzed));
06467             kmax = VMIN2(nz-1, (int)floor((pos[2] + zspan)/hzed));
06468             for (k=kmin; k<=kmax; k++) {
06469                 z = hzed*k;
06470                 dz2 = VSQR(pos[2] - z);

```

```

06471
06472     r2 = dx2 + dy2 + dz2;
06473
06474     /* We need to determine the inclusion value a000 at (i,j,k) */
06475     if (r2 < rtot2) a000 = 1.0;
06476     else a000 = 0.0;
06477
06478     /* We need to evaluate the values of x which intersect the
06479      * sphere and determine if these are in the interval
06480      * [(i,j,k), (i+1,j,k)] */
06481     if (r2 < (rtot2 - hx*hx)) a100 = 1.0;
06482     else if (r2 > (rtot2 + hx*hx)) a100 = 0.0;
06483     else if (rtot2 > (dy2 + dz2)) {
06484         dx = VSQRT(rtot2 - dy2 - dz2);
06485         xm = pos[0] - dx;
06486         xp = pos[0] + dx;
06487         if ((xm < x+hx) && (xm > x)) {
06488             a100 = (xm - x)/hx;
06489         } else if ((xp < x+hx) && (xp > x)) {
06490             a100 = (xp - x)/hx;
06491         }
06492     } else a100 = 0.0;
06493
06494     /* We need to evaluate the values of y which intersect the
06495      * sphere and determine if these are in the interval
06496      * [(i,j,k), (i,j+1,k)] */
06497     if (r2 < (rtot2 - hy*hy)) a010 = 1.0;
06498     else if (r2 > (rtot2 + hy*hy)) a010 = 0.0;
06499     else if (rtot2 > (dx2 + dz2)) {
06500         dy = VSQRT(rtot2 - dx2 - dz2);
06501         ym = pos[1] - dy;
06502         yp = pos[1] + dy;
06503         if ((ym < y+hy) && (ym > y)) {
06504             a010 = (ym - y)/hy;
06505         } else if ((yp < y+hy) && (yp > y)) {
06506             a010 = (yp - y)/hy;
06507         }
06508     } else a010 = 0.0;
06509
06510     /* We need to evaluate the values of z which intersect the
06511      * sphere and determine if these are in the interval
06512      * [(i,j,k), (i,j,k+1)] */
06513     if (r2 < (rtot2 - hzed*hzed)) a001 = 1.0;
06514     else if (r2 > (rtot2 + hzed*hzed)) a001 = 0.0;
06515     else if (rtot2 > (dx2 + dy2)) {
06516         dz = VSQRT(rtot2 - dx2 - dy2);
06517         zm = pos[2] - dz;
06518         zp = pos[2] + dz;
06519         if ((zm < z+hzed) && (zm > z)) {
06520             a001 = (zm - z)/hzed;
06521         } else if ((zp < z+hzed) && (zp > z)) {
06522             a001 = (zp - z)/hzed;
06523         }
06524     } else a001 = 0.0;
06525
06526     if (a100 < xarray[IJK(i,j,k)]) xarray[IJK(i,j,k)] = a100;
06527     if (a010 < yarray[IJK(i,j,k)]) yarray[IJK(i,j,k)] = a010;

```

```

06528         if (a001 < zarray[IJK(i,j,k)]) zarray[IJK(i,j,k)] = a001;
06529
06530     } /* k loop */
06531 } /* j loop */
06532 } /* i loop */
06533 }
06534
06535 /*
06536
06537 NOTE: This is the original version of the markSphere function. It's in here
06538 for reference and in case a reversion to the original code is needed.
06539 D. Gohara (2/14/08)
06540 */
06541 /*
06542 VPRIVATE void markSphere(
06543     double rtot, double *tpos,
06544     int nx, int ny, int nz,
06545     double hx, double hy, double hzed,
06546     double xmin, double ymin, double zmin,
06547     double *array, double markVal) {
06548
06549     int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
06550     double dx, dx2, dy, dy2, dz, dz2;
06551     double rtot2, pos[3];
06552
06553     // Convert to grid reference frame
06554     pos[0] = tpos[0] - xmin;
06555     pos[1] = tpos[1] - ymin;
06556     pos[2] = tpos[2] - zmin;
06557
06558     rtot2 = VSQR(rtot);
06559
06560     dx = rtot + 0.5*hx;
06561     imin = VMAX2(0,(int)ceil((pos[0] - dx)/hx));
06562     imax = VMIN2(nx-1,(int)floor((pos[0] + dx)/hx));
06563     for (i=imin; i<=imax; i++) {
06564         dx2 = VSQR(pos[0] - hx*i);
06565         if (rtot2 > dx2) {
06566             dy = VSQRT(rtot2 - dx2) + 0.5*hy;
06567         } else {
06568             dy = 0.5*hy;
06569         }
06570         jmin = VMAX2(0,(int)ceil((pos[1] - dy)/hy));
06571         jmax = VMIN2(ny-1,(int)floor((pos[1] + dy)/hy));
06572         for (j=jmin; j<=jmax; j++) {
06573             dy2 = VSQR(pos[1] - hy*j);
06574             if (rtot2 > (dx2+dy2)) {
06575                 dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
06576             } else {
06577                 dz = 0.5*hzed;
06578             }
06579             kmin = VMAX2(0,(int)ceil((pos[2] - dz)/hzed));
06580             kmax = VMIN2(nz-1,(int)floor((pos[2] + dz)/hzed));
06581             for (k=kmin; k<=kmax; k++) {
06582                 dz2 = VSQR(k*hzed - pos[2]);
06583                 if ((dz2 + dy2 + dx2) <= rtot2) {
06584                     array[IJK(i,j,k)] = markVal;

```

```

06585     }
06586   } // k loop
06587 } // j loop
06588 } // i loop
06589 }
06590 */
06591 VPRIvATE void markSphere(double rtot, double *tpos,
06592     int nx, int ny, int nz,
06593     double hx, double hy, double hz,
06594     double xmin, double ymin, double zmin,
06595     double *array, double markVal) {
06596
06597     int i, j, k;
06598     double fi,fj,fk;
06599     int imin, imax;
06600     int jmin, jmax;
06601     int kmin, kmax;
06602     double dx2, dy2, dz2;
06603     double xrange, yrange, zrange;
06604     double rtot2, posx, posy, posz;
06605
06606     /* Convert to grid reference frame */
06607     posx = tpos[0] - xmin;
06608     posy = tpos[1] - ymin;
06609     posz = tpos[2] - zmin;
06610
06611     rtot2 = VSQR(rtot);
06612
06613     xrange = rtot + 0.5 * hx;
06614     yrange = rtot + 0.5 * hy;
06615     zrange = rtot + 0.5 * hz;
06616
06617     imin = VMAX2(0, (int)ceil((posx - xrange)/hx));
06618     jmin = VMAX2(0, (int)ceil((posy - yrange)/hy));
06619     kmin = VMAX2(0, (int)ceil((posz - zrange)/hz));
06620
06621     imax = VMIN2(nx-1, (int)floor((posx + xrange)/hx));
06622     jmax = VMIN2(ny-1, (int)floor((posy + yrange)/hy));
06623     kmax = VMIN2(nz-1, (int)floor((posz + zrange)/hz));
06624
06625     for (i=imin, fi=imin; i<=imax; i++, fi+=1.) {
06626         dx2 = VSQR(posx - hx*fi);
06627         for (j=jmin, fj=jmin; j<=jmax; j++, fj+=1.) {
06628             dy2 = VSQR(posy - hy*fj);
06629             if((dx2 + dy2) > rtot2) continue;
06630             for (k=kmin, fk=kmin; k<=kmax; k++, fk+=1.) {
06631                 dz2 = VSQR(posz - hz*fk);
06632                 if ((dz2 + dy2 + dx2) <= rtot2) {
06633                     array[IJK(i, j, k)] = markVal;
06634                 }
06635             }
06636         }
06637     }
06638 }
06639
06640 VPRIvATE void zlapSolve(
06641     Vpmg *thee,

```

```

06642     double **solution,
06643     double **source,
06644     double **work1
06645     ) {
06646
06647     /* NOTE: this is an incredibly inefficient algorithm. The next
06648      * improvement is to focus on only non-zero entries in the source term.
06649      * The best improvement is to use a fast sine transform */
06650
06651     int n, nx, ny, nz, i, j, k, kx, ky, kz;
06652     double hx, hy, hzed, wx, wy, wz, xlen, ylen, zlen;
06653     double phix, phixp1, phixml, phiy, phiyml, phiyp1, phiz, phizml, phizp1;
06654     double norm, coef, proj, eigx, eigy, eigz;
06655     double ihx2, ihy2, ihzed2;
06656     double *u, *f, *phi;
06657
06658     /* Snarf grid parameters */
06659     nx = thee->pmgp->nx;
06660     ny = thee->pmgp->ny;
06661     nz = thee->pmgp->nz;
06662     n = nx*ny*nz;
06663     hx = thee->pmgp->hx;
06664     ihx2 = 1.0/hx/hx;
06665     hy = thee->pmgp->hy;
06666     ihy2 = 1.0/hy/hy;
06667     hzed = thee->pmgp->hzed;
06668     ihzed2 = 1.0/hzed/hzed;
06669     xlen = thee->pmgp->xlen;
06670     ylen = thee->pmgp->ylen;
06671     zlen = thee->pmgp->zlen;
06672
06673     /* Set solution and source array pointers */
06674     u = *solution;
06675     f = *source;
06676     phi = *work1;
06677
06678     /* Zero out the solution vector */
06679     for (i=0; i<n; i++) thee->u[i] = 0.0;
06680
06681     /* Iterate through the wavenumbers */
06682     for (kx=1; kx<(nx-1); kx++) {
06683
06684         wx = (VPI*(double)kx)/((double)nx - 1.0);
06685         eigx = 2.0*ihx2*(1.0 - cos(wx));
06686
06687         for (ky=1; ky<(ny-1); ky++) {
06688
06689             wy = (VPI*(double)ky)/((double)ny - 1.0);
06690             eigy = 2.0*ihy2*(1.0 - cos(wy));
06691
06692             for (kz=1; kz<(nz-1); kz++) {
06693
06694                 wz = (VPI*(double)kz)/((double)nz - 1.0);
06695                 eigz = 2.0*ihzed2*(1.0 - cos(wz));
06696
06697                 /* Calculate the basis function.
06698                  * We could calculate each basis function as

```

```

06699     *    phix(i) = sin(wx*i)
06700     *    phiy(j) = sin(wy*j)
06701     *    phiz(k) = sin(wz*k)
06702     * However, this is likely to be very expensive.
06703     * Therefore, we can use the fact that
06704     *    phix(i+1) = (2-hx*hx*eigx)*phix(i) - phix(i-1)
06705     * */
06706 for (i=1; i<(nx-1); i++) {
06707     if (i == 1) {
06708         phix = sin(wx*(double)i);
06709         phixml = 0.0;
06710     } else {
06711         phixp1 = (2.0-hx*hx*eigx)*phix - phixml;
06712         phixml = phix;
06713         phix = phixp1;
06714     }
06715     /* phix = sin(wx*(double)i); */
06716     for (j=1; j<(ny-1); j++) {
06717         if (j == 1) {
06718             phiy = sin(wy*(double)j);
06719             phiym1 = 0.0;
06720         } else {
06721             phiyp1 = (2.0-hy*hy*eigy)*phiy - phiym1;
06722             phiym1 = phiy;
06723             phiy = phiyp1;
06724         }
06725         /* phiy = sin(wy*(double)j); */
06726         for (k=1; k<(nz-1); k++) {
06727             if (k == 1) {
06728                 phiz = sin(wz*(double)k);
06729                 phizml = 0.0;
06730             } else {
06731                 phizp1 = (2.0-hzed*hzed*eigz)*phiz - phizml;
06732                 phizml = phiz;
06733                 phiz = phizp1;
06734             }
06735             /* phiz = sin(wz*(double)k); */
06736
06737             phi[IJK(i,j,k)] = phix*phiy*phiz;
06738
06739         }
06740     }
06741 }
06742
06743 /* Calculate the projection of the source function on this
06744 * basis function */
06745 proj = 0.0;
06746 for (i=1; i<(nx-1); i++) {
06747     for (j=1; j<(ny-1); j++) {
06748         for (k=1; k<(nz-1); k++) {
06749
06750             proj += f[IJK(i,j,k)]*phi[IJK(i,j,k)];
06751
06752         } /* k loop */
06753     } /* j loop */
06754 } /* i loop */
06755

```

```

06756             /* Assemble the coefficient to weight the contribution of this
06757             * basis function to the solution */
06758             /* The first contribution is the projection */
06759             coef = proj;
06760             /* The second contribution is the eigenvalue */
06761             coef = coef/(eigx + eigy + eigz);
06762             /* The third contribution is the normalization factor */
06763             coef = (8.0/xlen/ylen/zlen)*coef;
06764             /* The fourth contribution is from scaling the diagonal */
06765             /* coef = hx*hy*hzed*coef; */
06766
06767             /* Evaluate the basis function at each grid point */
06768             for (i=1; i<(nx-1); i++) {
06769                 for (j=1; j<(ny-1); j++) {
06770                     for (k=1; k<(nz-1); k++) {
06771
06772                         u[IJK(i,j,k)] += coef*phi[IJK(i,j,k)];
06773
06774                     } /* k loop */
06775                 } /* j loop */
06776             } /* i loop */
06777
06778         } /* kz loop */
06779     } /* ky loop */
06780 } /* kx loop */
06781
06782 }
06783
06784 VPUBLIC int Vpmg_solveLaplace(Vpmg *thee) {
06785
06786     int i, j, k, ijk, nx, ny, nz, n, dilo, dihi, djlo, djhi, dklo, dkhi;
06787     double hx, hy, hzed, epsw, iepsw, scal, scalx, scaly, scalz;
06788
06789     nx = thee->pmgp->nx;
06790     ny = thee->pmgp->ny;
06791     nz = thee->pmgp->nz;
06792     n = nx*ny*nz;
06793     hx = thee->pmgp->hx;
06794     hy = thee->pmgp->hy;
06795     hzed = thee->pmgp->hzed;
06796     epsw = Vpbe_getSolventDiel(thee->pbe);
06797     iepsw = 1.0/epsw;
06798     scal = hx*hy*hzed;
06799     scalx = hx*hy/hzed;
06800     scaly = hx*hzed/hy;
06801     scalz = hx*hy/hzed;
06802
06803     if (!(thee->filled)) {
06804         Vnm_print(2, "Vpmg_solve: Need to call Vpmg_fillco()!\n");
06805         return 0;
06806     }
06807
06808     /* Load boundary conditions into the RHS array */
06809     for (i=1; i<(nx-1); i++) {
06810
06811         if (i == 1) dilo = 1;
06812         else dilo = 0;

```

```

06813     if (i == nx-2) dihi = 1;
06814     else dihi = 0;
06815
06816     for (j=1; j<(ny-1); j++) {
06817         if (j == 1) djlo = 1;
06818         else djlo = 0;
06819         if (j == ny-2) djhi = 1;
06820         else djhi = 0;
06821
06822         for (k=1; k<(nz-1); k++) {
06823             if (k == 1) dklo = 1;
06824             else dklo = 0;
06825             if (k == nz-2) dkhi = 1;
06826             else dkhi = 0;
06827
06828             thee->fcf[IJK(i,j,k)] = \
06829                 iepsw*scalx*thee->charge[IJK(i,j,k)] \
06830                 + dilo*scalx*thee->gxfc[IJKx(j,k,0)] \
06831                 + dihi*scalx*thee->gxfc[IJKx(j,k,1)] \
06832                 + djlo*scaly*thee->gycf[IJKy(i,k,0)] \
06833                 + djhi*scaly*thee->gycf[IJKy(i,k,1)] \
06834                 + dklo*scalz*thee->gzcf[IJKz(i,j,0)] \
06835                 + dkhi*scalz*thee->gzcf[IJKz(i,j,1)] ;
06836
06837
06838
06839     }
06840   }
06841 }
06842
06843 /* Solve */
06844 zlapSolve( thee, &(thee->u), &(thee->fcf), &(thee->tcf) );
06845
06846 /* Add boundary conditions to solution */
06847 /* i faces */
06848 for (j=0; j<ny; j++) {
06849   for (k=0; k<nz; k++) {
06850     thee->u[IJK(0,j,k)] = thee->gxfc[IJKx(j,k,0)];
06851     thee->u[IJK(nx-1,j,k)] = thee->gycf[IJKx(j,k,1)];
06852   }
06853 }
06854 /* j faces */
06855 for (i=0; i<nx; i++) {
06856   for (k=0; k<nz; k++) {
06857     thee->u[IJK(i,0,k)] = thee->gycf[IJKy(i,k,0)];
06858     thee->u[IJK(i,ny-1,k)] = thee->gycf[IJKy(i,k,1)];
06859   }
06860 }
06861 /* k faces */
06862 for (i=0; i<nx; i++) {
06863   for (j=0; j<ny; j++) {
06864     thee->u[IJK(i,j,0)] = thee->gzcf[IJKz(i,j,0)];
06865     thee->u[IJK(i,j,nz-1)] = thee->gzcf[IJKz(i,j,1)];
06866   }
06867 }
06868
06869 return 1;

```

```
06870
06871 }
06872
06873 VPRIATE double VFCHI4(int i, double f) {
06874     return (2.5+((double)(i)-(f)));
06875 }
06876
06877 VPRIATE double bspline4(double x) {
06878
06879     double m, m2;
06880     static double one6 = 1.0/6.0;
06881     static double one8 = 1.0/8.0;
06882     static double one24 = 1.0/24.0;
06883     static double thirteen24 = 13.0/24.0;
06884     static double fortyseven24 = 47.0/24.0;
06885     static double seventeen24 = 17.0/24.0;
06886
06887     if ((x > 0.0) && (x <= 1.0)){
06888         m = x*x;
06889         return one24*m*m;
06890     } else if ((x > 1.0) && (x <= 2.0)){
06891         m = x - 1.0;
06892         m2 = m*m;
06893         return -one8 + one6*x + m2*(0.25 + one6*m - one6*m2);
06894     } else if ((x > 2.0) && (x <= 3.0)){
06895         m = x - 2.0;
06896         m2 = m*m;
06897         return -thirteen24 + 0.5*x + m2*(-0.25 - 0.5*m + 0.25*m2);
06898     } else if ((x > 3.0) && (x <= 4.0)){
06899         m = x - 3.0;
06900         m2 = m*m;
06901         return fortyseven24 - 0.5*x + m2*(-0.25 + 0.5*m - one6*m2);
06902     } else if ((x > 4.0) && (x <= 5.0)){
06903         m = x - 4.0;
06904         m2 = m*m;
06905         return seventeen24 - one6*x + m2*(0.25 - one6*m + one24*m2);
06906     } else {
06907         return 0.0;
06908     }
06909 }
06910
06911 VPUBLIC double dbspline4(double x) {
06912
06913     double m, m2;
06914     static double one6 = 1.0/6.0;
06915     static double one3 = 1.0/3.0;
06916     static double two3 = 2.0/3.0;
06917     static double thirteen6 = 13.0/6.0;
06918
06919     if ((x > 0.0) && (x <= 1.0)){
06920         m2 = x*x;
06921         return one6*x*m2;
06922     } else if ((x > 1.0) && (x <= 2.0)){
06923         m = x - 1.0;
06924         m2 = m*m;
06925         return -one3 + 0.5*x + m2*(0.5 - two3*m);
06926     } else if ((x > 2.0) && (x <= 3.0)) {
```

```

06927     m = x - 2.0;
06928     m2 = m*m;
06929     return 1.5 - 0.5*x + m2*(-1.5 + m);
06930 } else if ((x > 3.0) && (x <= 4.0)) {
06931     m = x - 3.0;
06932     m2 = m*m;
06933     return 1.0 - 0.5*x + m2*(1.5 - two3*m);
06934 } else if ((x > 4.0) && (x <= 5.0)) {
06935     m = x - 4.0;
06936     m2 = m*m;
06937     return -thirteen6 + 0.5*x + m2*(-0.5 + one6*m);
06938 } else {
06939     return 0.0;
06940 }
06941 }
06942
06943 VPUBLIC double d2bspline4(double x) {
06944
06945     double m, m2;
06946
06947     if ((x > 0.0) && (x <= 1.0)) {
06948         return 0.5*x*x;
06949     } else if ((x > 1.0) && (x <= 2.0)) {
06950         m = x - 1.0;
06951         m2 = m*m;
06952         return -0.5 + x - 2.0*m2;
06953     } else if ((x > 2.0) && (x <= 3.0)) {
06954         m = x - 2.0;
06955         m2 = m*m;
06956         return 5.5 - 3.0*x + 3.0*m2;
06957     } else if ((x > 3.0) && (x <= 4.0)) {
06958         m = x - 3.0;
06959         m2 = m*m;
06960         return -9.5 + 3.0*x - 2.0*m2;
06961     } else if ((x > 4.0) && (x <= 5.0)) {
06962         m = x - 4.0;
06963         m2 = m*m;
06964         return 4.5 - x + 0.5*m2;
06965     } else {
06966         return 0.0;
06967     }
06968 }
06969
06970 VPUBLIC double d3bspline4(double x) {
06971
06972     if ((x > 0.0) && (x <= 1.0)) return x;
06973     else if ((x > 1.0) && (x <= 2.0)) return 5.0 - 4.0 * x;
06974     else if ((x > 2.0) && (x <= 3.0)) return -15.0 + 6.0 * x;
06975     else if ((x > 3.0) && (x <= 4.0)) return 15.0 - 4.0 * x;
06976     else if ((x > 4.0) && (x <= 5.0)) return x - 5.0;
06977     else return 0.0;
06978 }
06979 }
06980
06981 VPUBLIC void fillcoPermanentMultipole(Vpmg *thee) {
06982
06983     Valist *alist;

```

```

06984     Vpbe *pbe;
06985     Vatom *atom;
06986     /* Coversions */
06987     double zmagic, f;
06988     /* Grid */
06989     double xmin, xmax, ymin, ymax, zmin, zmax;
06990     double xlabel, ylabel, zlabel, position[3], ifloat, jfloat, kfloat;
06991     double hx, hy, hzed, *apos;
06992     /* Multipole */
06993     double charge, *dipole, *quad;
06994     double c, ux, uy, uz, qx, qy, qz, qxx, qyy, qzx, qzy, qzz, qave;
06995     /* B-spline weights */
06996     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz;
06997     double mi, mj, mk;
06998     /* Loop variables */
06999     int i, ii, jj, kk, nx, ny, nz, iatom;
07000     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07001     /* sanity check */
07002     double mir, mjr, mkr, mr2;
07004     double debye, mc, mux, muy, muz, mqxx, mqyx, mqyy, mqzx, mqzy, mqzz;
07005
07006     VASSERT(thee != VNULL);
07007
07008     /* Get PBE info */
07009     pbe = thee->pbe;
07010     alist = pbe->alist;
07011     zmagic = Vpbe_getZmagic(pbe);
07012
07013     /* Mesh info */
07014     nx = thee->pmgp->nx;
07015     ny = thee->pmgp->ny;
07016     nz = thee->pmgp->nz;
07017     hx = thee->pmgp->hx;
07018     hy = thee->pmgp->hy;
07019     hzed = thee->pmgp->hzed;
07020
07021     /* Conversion */
07022     f = zmagic/(hx*hy*hzed);
07023
07024     /* Define the total domain size */
07025     xlabel = thee->pmgp->xlabel;
07026     ylabel = thee->pmgp->ylabel;
07027     zlabel = thee->pmgp->zlabel;
07028
07029     /* Define the min/max dimensions */
07030     xmin = thee->pmgp->xcent - (xlabel/2.0);
07031     ymin = thee->pmgp->ycent - (ylabel/2.0);
07032     zmin = thee->pmgp->zcent - (zlabel/2.0);
07033     xmax = thee->pmgp->xcent + (xlabel/2.0);
07034     ymax = thee->pmgp->ycent + (ylabel/2.0);
07035     zmax = thee->pmgp->zcent + (zlabel/2.0);
07036
07037     /* Fill in the source term (permanent atomic multipoles) */
07038     Vnm_print(0, "fillcoPermanentMultipole: filling in source term.\n");
07039     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07040

```

```

07041     atom = Valist_getAtom(alist, iatom);
07042     apos = Vatom_getPosition(atom);
07043
07044     c = Vatom_getCharge(atom)*f;
07045
07046 #if defined(WITH_TINKER)
07047     dipole = Vatom_getDipole(atom);
07048     ux = dipole[0]/hx*f;
07049     uy = dipole[1]/hy*f;
07050     uz = dipole[2]/hzed*f;
07051     quad = Vatom_getQuadrupole(atom);
07052     qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
07053     qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
07054     qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
07055     qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
07056     qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
07057     qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
07058 #else
07059     ux = 0.0;
07060     uy = 0.0;
07061     uz = 0.0;
07062     qxx = 0.0;
07063     qyx = 0.0;
07064     qyy = 0.0;
07065     qzx = 0.0;
07066     qzy = 0.0;
07067     qzz = 0.0;
07068 #endif /* if defined(WITH_TINKER) */
07069
07070     /* check
07071     mc = 0.0;
07072     mux = 0.0;
07073     muy = 0.0;
07074     muz = 0.0;
07075     mqxx = 0.0;
07076     mqyx = 0.0;
07077     mqyy = 0.0;
07078     mqzx = 0.0;
07079     mqzy = 0.0;
07080     mqzz = 0.0; */
07081
07082     /* Make sure we're on the grid */
07083     if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
07084         (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
07085         (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07086         Vnm_print(2, "fillcoPermanentMultipole: Atom #%d at (%4.3f, %4.3f, %4
.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
07087
07088         Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin
, xmax);
07089         Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin
, ymax);
07090         Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin
, zmax);
07091         fflush(stderr);
07092     } else {
07093

```

```

07093     /* Convert the atom position to grid reference frame */
07094     position[0] = apos[0] - xmin;
07095     position[1] = apos[1] - ymin;
07096     position[2] = apos[2] - zmin;
07097
07098     /* Figure out which vertices we're next to */
07099     ifloat = position[0]/hx;
07100     jfloat = position[1]/hy;
07101     kfloat = position[2]/hzed;
07102
07103     ip1    = (int)ceil(ifloat);
07104     ip2    = ip1 + 2;
07105     im1    = (int)floor(ifloat);
07106     im2    = im1 - 2;
07107     jp1    = (int)ceil(jfloat);
07108     jp2    = jp1 + 2;
07109     jm1    = (int)floor(jfloat);
07110     jm2    = jm1 - 2;
07111     kp1    = (int)ceil(kfloat);
07112     kp2    = kp1 + 2;
07113     km1    = (int)floor(kfloat);
07114     km2    = km1 - 2;
07115
07116     /* This step shouldn't be necessary, but it saves nasty debugging
07117      * later on if something goes wrong */
07118     ip2 = VMIN2(ip2,nx-1);
07119     ip1 = VMIN2(ip1,nx-1);
07120     im1 = VMAX2(im1,0);
07121     im2 = VMAX2(im2,0);
07122     jp2 = VMIN2(jp2,ny-1);
07123     jp1 = VMIN2(jp1,ny-1);
07124     jm1 = VMAX2(jm1,0);
07125     jm2 = VMAX2(jm2,0);
07126     kp2 = VMIN2(kp2,nz-1);
07127     kp1 = VMIN2(kp1,nz-1);
07128     km1 = VMAX2(km1,0);
07129     km2 = VMAX2(km2,0);
07130
07131     /* Now assign fractions of the charge to the nearby verts */
07132     for (ii=im2; ii<=ip2; ii++) {
07133         mi = VFCHI4(ii,ifloat);
07134         mx = bspline4(mi);
07135         dmx = dbspline4(mi);
07136         d2mx = d2bspline4(mi);
07137         for (jj=jm2; jj<=jp2; jj++) {
07138             mj = VFCHI4(jj,jfloat);
07139             my = bspline4(mj);
07140             dmy = dbspline4(mj);
07141             d2my = d2bspline4(mj);
07142             for (kk=km2; kk<=kp2; kk++) {
07143                 mk = VFCHI4(kk,kfloat);
07144                 mz = bspline4(mk);
07145                 dmz = dbspline4(mk);
07146                 d2mz = d2bspline4(mk);
07147                 charge = mx*my*mz*c -
07148                         dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
07149                         d2mx*my*mz*qxx +

```

```

07150      dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
07151      dmz*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
07152      thee->charge[IJK(ii,jj,kk)] += charge;
07153
07154      /* sanity check - recalculate traceless multipoles
07155      from the grid charge distribution for this
07156      site.
07157
07158      mir = (mi - 2.5) * hx;
07159      mjr = (mj - 2.5) * hy;
07160      mkr = (mk - 2.5) * hzed;
07161      mr2 = mir*mir+mjr*mjr+mkr*mkr;
07162      mc += charge;
07163      mux += mir * charge;
07164      muy += mjr * charge;
07165      muz += mkr * charge;
07166      mqxx += (1.5*mir*mir - 0.5*mr2) * charge;
07167      mqyx += 1.5*mjr*mir * charge;
07168      mqyy += (1.5*mjr*mjr - 0.5*mr2) * charge;
07169      mqzx += 1.5*mkr*mir * charge;
07170      mqzy += 1.5*mkr*mjr * charge;
07171      mqzz += (1.5*mkr*mkr - 0.5*mr2) * charge;
07172      */
07173      }
07174      }
07175      }
07176  } /* endif (on the mesh) */
07177
07178  /* print out the Grid vs. Ideal Point Multipole. */
07179
07180  /*
07181  debye = 4.8033324;
07182  mc = mc/f;
07183  mux = mux/f*debye;
07184  muy = muy/f*debye;
07185  muz = muz/f*debye;
07186  mqxx = mqxx/f*debye;
07187  mqyy = mqyy/f*debye;
07188  mqzz = mqzz/f*debye;
07189  mqyx = mqyx/f*debye;
07190  mqzx = mqzx/f*debye;
07191  mqzy = mqzy/f*debye;
07192
07193  printf(" Grid v. Actual Permanent Multipole for Site %i\n",iatom);
07194  printf(" G: %10.6f\n",mc);
07195  printf(" A: %10.6f\n",c/f);
07196  printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07197  printf(" A: %10.6f %10.6f %10.6f\n",
07198      (ux * hx / f) * debye,
07199      (uy * hy / f) * debye,
07200      (uz * hzed / f) * debye);
07201  printf(" G: %10.6f\n",mqxx);
07202  printf(" A: %10.6f\n",quad[0]*debye);
07203  printf(" G: %10.6f %10.6f\n",mqyx,mqyy);
07204  printf(" A: %10.6f %10.6f\n",quad[3]*debye,quad[4]*debye);
07205  printf(" G: %10.6f %10.6f %10.6f\n",mqzx,mqzy,mqzz);
07206  printf(" A: %10.6f %10.6f %10.6f\n",

```

```

07207             quad[6]*debye,quad[7]*debye,quad[8]*debye); /*/
07208
07209 } /* endfor (each atom) */
07210 }
07211
07212 #if defined(WITH_TINKER)
07213
07214 VPUBLIC void fillcoInducedDipole(Vpmg *thee) {
07215
07216     Valist *alist;
07217     Vpbe *pbe;
07218     Vatom *atom;
07219     /* Conversions */
07220     double zmagic, f;
07221     /* Grid */
07222     double xmin, xmax, ymin, ymax, zmin, zmax;
07223     double xlabel, ylabel, zlabel, ifloat, jfloat, kfloat;
07224     double hx, hy, hzed, *apos, position[3];
07225     /* B-spline weights */
07226     double mx, my, mz, dmx, dmy, dmz;
07227     /* Dipole */
07228     double charge, *dipole, ux, uy, uz;
07229     double mi, mj, mk;
07230     /* Loop indeces */
07231     int i, ii, jj, nx, ny, nz, iatom;
07232     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07233
07234     double debye;
07235     double mux, muy, muz;
07236     double mir, mjr, mkr;
07237
07238     VASSERT(thee != VNULL);
07239
07240     /* Get PBE info */
07241     pbe = thee->pbe;
07242     alist = pbe->alist;
07243     zmagic = Vpbe_getZmagic(pbe);
07244
07245     /* Mesh info */
07246     nx = thee->pmgp->nx;
07247     ny = thee->pmgp->ny;
07248     nz = thee->pmgp->nz;
07249     hx = thee->pmgp->hx;
07250     hy = thee->pmgp->hy;
07251     hzed = thee->pmgp->hzed;
07252
07253     /* Conversion */
07254     f = zmagic/(hx*hy*hzed);
07255
07256     /* Define the total domain size */
07257     xlabel = thee->pmgp->xlabel;
07258     ylabel = thee->pmgp->ylabel;
07259     zlabel = thee->pmgp->zlabel;
07260
07261     /* Define the min/max dimensions */
07262     xmin = thee->pmgp->xcent - (xlabel/2.0);
07263     ymin = thee->pmgp->ycent - (ylabel/2.0);

```

```

07264     zmin = thee->pmgp->zcent - (zlen/2.0);
07265     xmax = thee->pmgp->xcent + (xlen/2.0);
07266     ymax = thee->pmgp->ycent + (ylen/2.0);
07267     zmax = thee->pmgp->zcent + (zlen/2.0);
07268
07269     /* Fill in the source term (induced dipoles) */
07270     Vnm_print(0, "fillcoInducedDipole: filling in the source term.\n");
07271     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07272
07273         atom = Valist_getAtom(alist, iatom);
07274         apos = Vatom_getPosition(atom);
07275
07276         dipole = Vatom_getInducedDipole(atom);
07277         ux = dipole[0]/hx*f;
07278         uy = dipole[1]/hy*f;
07279         uz = dipole[2]/hzed*f;
07280
07281         mux = 0.0;
07282         muy = 0.0;
07283         muz = 0.0;
07284
07285         /* Make sure we're on the grid */
07286         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
07287             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
07288             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07289             Vnm_print(2, "fillcoInducedDipole: Atom #%"PRIu32" at (%4.3f, %4.3f, %4.3f)\n"
07290             "is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
07291             Vnm_print(2, "fillcoInducedDipole: xmin = %g, xmax = %g\n", xmin, xmax);
07292             Vnm_print(2, "fillcoInducedDipole: ymin = %g, ymax = %g\n", ymin, ymax);
07293             Vnm_print(2, "fillcoInducedDipole: zmin = %g, zmax = %g\n", zmin, zmax);
07294             fflush(stderr);
07295         } else {
07296
07297             /* Convert the atom position to grid reference frame */
07298             position[0] = apos[0] - xmin;
07299             position[1] = apos[1] - ymin;
07300             position[2] = apos[2] - zmin;
07301
07302             /* Figure out which vertices we're next to */
07303             ifloat = position[0]/hx;
07304             jfloat = position[1]/hy;
07305             kfloat = position[2]/hzed;
07306
07307             ip1 = (int)ceil(ifloat);
07308             ip2 = ip1 + 2;
07309             im1 = (int)floor(ifloat);
07310             im2 = im1 - 2;
07311             jp1 = (int)ceil(jfloat);
07312             jp2 = jp1 + 2;
07313             jm1 = (int)floor(jfloat);
07314             jm2 = jm1 - 2;
07315             kp1 = (int)ceil(kfloat);
07316             kp2 = kp1 + 2;
07317             km1 = (int)floor(kfloat);

```

```

07317         km2 = km1 - 2;
07318
07319     /* This step shouldn't be necessary, but it saves nasty debugging
07320      * later on if something goes wrong */
07321     ip2 = VMIN2(ip2,nx-1);
07322     ip1 = VMIN2(ip1,nx-1);
07323     im1 = VMAX2(im1,0);
07324     im2 = VMAX2(im2,0);
07325     jp2 = VMIN2(jp2,ny-1);
07326     jp1 = VMIN2(jp1,ny-1);
07327     jm1 = VMAX2(jm1,0);
07328     jm2 = VMAX2(jm2,0);
07329     kp2 = VMIN2(kp2,nz-1);
07330     kp1 = VMIN2(kp1,nz-1);
07331     km1 = VMAX2(km1,0);
07332     km2 = VMAX2(km2,0);
07333
07334     /* Now assign fractions of the dipole to the nearby verts */
07335     for (ii=im2; ii<=ip2; ii++) {
07336         mi = VFCHI4(ii,ifloat);
07337         mx = bspline4(mi);
07338         dmx = dbspline4(mi);
07339         for (jj=jm2; jj<=jp2; jj++) {
07340             mj = VFCHI4(jj,jffloat);
07341             my = bspline4(mj);
07342             dmy = dbspline4(mj);
07343             for (kk=km2; kk<=kp2; kk++) {
07344                 mk = VFCHI4(kk,kffloat);
07345                 mz = bspline4(mk);
07346                 dmz = dbspline4(mk);
07347                 charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07348                 thee->charge[IJK(ii,jj,kk)] += charge;
07349
07350             /*
07351             mir = (mi - 2.5) * hx;
07352             mjr = (mj - 2.5) * hy;
07353             mkr = (mk - 2.5) * hzed;
07354             mux += mir * charge;
07355             muy += mjr * charge;
07356             muz += mkr * charge;
07357             */
07358         }
07359     }
07360 }
07361 } /* endif (on the mesh) */
07362
07363 /* check
07364 debye = 4.8033324;
07365 mux = mux/f*debye;
07366 muy = muy/f*debye;
07367 muz = muz/f*debye;
07368
07369 printf(" Grid v. Actual Induced Dipole for Site %i\n",iatom);
07370 printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07371 printf(" A: %10.6f %10.6f %10.6f\n",
07372           (ux * hx / f) * debye,
07373           (uy * hy / f) * debye,

```

```

07374             (uz * hzed /f) * debye);
07375         */
07376
07377     } /* endfor (each atom) */
07378 }
07379
07380 VPUBLIC void fillcoNLInducedDipole(Vpmg *thee) {
07381
07382     Valist *alist;
07383     Vpbe *pbe;
07384     Vatom *atom;
07385     /* Conversions */
07386     double zmagic, f;
07387     /* Grid */
07388     double xmin, xmax, ymin, ymax, zmin, zmax;
07389     double xlabel, ylabel, zlabel, ifloat, jfloat, kfloat;
07390     double hx, hy, hzed, *apos, position[3];
07391     /* B-spline weights */
07392     double mx, my, mz, dmx, dmy, dmz;
07393     /* Dipole */
07394     double charge, *dipole, ux, uy, uz;
07395     double mi, mj, mk;
07396     /* Loop indeces */
07397     int i, ii, jj, kk, nx, ny, nz, iatom;
07398     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07399
07400     /* sanity check
07401     double debye;
07402     double mux, muy, muz;
07403     double mir, mjr, mkr;
07404     */
07405
07406     VASSERT(thee != VNULL);
07407
07408     /* Get PBE info */
07409     pbe = thee->pbe;
07410     alist = pbe->alist;
07411     zmagic = Vpbe_getZmagic(pbe);
07412
07413     /* Mesh info */
07414     nx = thee->pmgp->nx;
07415     ny = thee->pmgp->ny;
07416     nz = thee->pmgp->nz;
07417     hx = thee->pmgp->hx;
07418     hy = thee->pmgp->hy;
07419     hzed = thee->pmgp->hzed;
07420
07421     /* Conversion */
07422     f = zmagic/(hx*hy*hzed);
07423
07424     /* Define the total domain size */
07425     xlabel = thee->pmgp->xlabel;
07426     ylabel = thee->pmgp->ylabel;
07427     zlabel = thee->pmgp->zlabel;
07428
07429     /* Define the min/max dimensions */
07430     xmin = thee->pmgp->xcent - (xlabel/2.0);

```

```

07431     ymin = thee->pmgp->ycent - (ylen/2.0);
07432     zmin = thee->pmgp->zcent - (zlen/2.0);
07433     xmax = thee->pmgp->xcent + (xlen/2.0);
07434     ymax = thee->pmgp->ycent + (ylen/2.0);
07435     zmax = thee->pmgp->zcent + (zlen/2.0);
07436
07437     /* Fill in the source term (non-local induced dipoles) */
07438     Vnm_print(0, "fillcoNLInducedDipole: filling in source term.\n");
07439     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07440
07441         atom = Valist_getAtom(alist, iatom);
07442         apos = Vatom_getPosition(atom);
07443
07444         dipole = Vatom_getNLInducedDipole(atom);
07445         ux = dipole[0]/hx*f;
07446         uy = dipole[1]/hy*f;
07447         uz = dipole[2]/hzed*f;
07448
07449         /*
07450             mux = 0.0;
07451             muy = 0.0;
07452             muz = 0.0;
07453             */
07454
07455         /* Make sure we're on the grid */
07456         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
07457             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
07458             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07459             Vnm_print(2, "fillcoNLInducedDipole: Atom #d at (%4.3f, %4.3f,%4.3f)\n"
07460             is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
07461             Vnm_print(2, "fillcoNLInducedDipole: xmin = %g, xmax = %g\n", xmin, x
07462             max);
07463             Vnm_print(2, "fillcoNLInducedDipole: ymin = %g, ymax = %g\n", ymin, y
07464             max);
07465             Vnm_print(2, "fillcoNLInducedDipole: zmin = %g, zmax = %g\n", zmin, z
07466             max);
07467             fflush(stderr);
07468         } else {
07469
07470             /* Convert the atom position to grid reference frame */
07471             position[0] = apos[0] - xmin;
07472             position[1] = apos[1] - ymin;
07473             position[2] = apos[2] - zmin;
07474
07475             /* Figure out which vertices we're next to */
07476             ifloat = position[0]/hx;
07477             jfloat = position[1]/hy;
07478             kfloat = position[2]/hzed;
07479
07480             ip1 = (int)ceil(ifloat);
07481             ip2 = ip1 + 2;
07482             im1 = (int)floor(ifloat);
07483             im2 = im1 - 2;
07484             jp1 = (int)ceil(jfloat);
07485             jp2 = jp1 + 2;
07486             jm1 = (int)floor(jfloat);
07487             jm2 = jm1 - 2;

```

```

07484     kp1    = (int)ceil(kffloat);
07485     kp2    = kp1 + 2;
07486     km1    = (int)floor(kffloat);
07487     km2    = km1 - 2;
07488
07489     /* This step shouldn't be necessary, but it saves nasty debugging
07490      * later on if something goes wrong */
07491     ip2 = VMIN2(ip2,nx-1);
07492     ip1 = VMIN2(ip1,nx-1);
07493     im1 = VMAX2(im1,0);
07494     im2 = VMAX2(im2,0);
07495     jp2 = VMIN2(jp2,ny-1);
07496     jp1 = VMIN2(jp1,ny-1);
07497     jm1 = VMAX2(jm1,0);
07498     jm2 = VMAX2(jm2,0);
07499     kp2 = VMIN2(kp2,nz-1);
07500     kp1 = VMIN2(kp1,nz-1);
07501     km1 = VMAX2(km1,0);
07502     km2 = VMAX2(km2,0);
07503
07504     /* Now assign fractions of the non local induced dipole
07505      to the nearby verts */
07506     for (ii=im2; ii<=ip2; ii++) {
07507         mi = VFCHI4(ii,ifloat);
07508         mx = bspline4(mi);
07509         dmx = dbspline4(mi);
07510         for (jj=jm2; jj<=jp2; jj++) {
07511             mj = VFCHI4(jj,jfloat);
07512             my = bspline4(mj);
07513             dmy = dbspline4(mj);
07514             for (kk=km2; kk<=kp2; kk++) {
07515                 mk = VFCHI4(kk,kffloat);
07516                 mz = bspline4(mk);
07517                 dmz = dbspline4(mk);
07518                 charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07519                 theee->charge[IJK(ii,jj,kk)] += charge;
07520
07521                 /*
07522                  mir = (mi - 2.5) * hx;
07523                  mjr = (mj - 2.5) * hy;
07524                  mkr = (mk - 2.5) * hzed;
07525                  mux += mir * charge;
07526                  muy += mjr * charge;
07527                  muz += mkr * charge;
07528                  */
07529             }
07530         }
07531     }
07532 } /* endif (on the mesh) */
07533
07534 /*
07535 debye = 4.8033324;
07536 mux = mux/f*debye;
07537 muy = muy/f*debye;
07538 muz = muz/f*debye;
07539
07540 printf(" Grid v. Actual Non-Local Induced Dipole for Site %i\n",iatom);

```

```

07541     printf(" G: %10.6f %10.6f %10.6f\n", mux, muy, muz);
07542     printf(" A: %10.6f %10.6f %10.6f\n\n",
07543             (ux * hx / f) * debye,
07544             (uy * hy / f) * debye,
07545             (uz * hzed / f) * debye); */
07546
07547 } /* endfor (each atom) */
07548 }
07549
07550 VPUBLIC double Vpmg_qfPermanentMultipoleEnergy(Vpmg *thee, int atomID) {
07551
07552     double *u;
07553     Vatom *atom;
07554     /* Grid variables */
07555     int nx, ny, nz;
07556     double xmax, xmin, ymax, ymin, zmax, zmin;
07557     double hx, hy, hzed, ifloat, jfloat, kfloat;
07558     double mi, mj, mk;
07559     double *position;
07560     /* B-spline weights */
07561     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz;
07562     /* Loop indeces */
07563     int ip1, ip2, im1, im2, jp1, jp2, jm1, jm2, kp1, kp2, km1, km2;
07564     int i, j, ii, jj, kk;
07565     /* Potential, field, field gradient and multipole components */
07566     double pot, rfe[3], rfd[e[3][3]], energy;
07567     double f, charge, *dipole, *quad;
07568     double qxx, qyx, qyy, qzx, qzy, qzz;
07569
07570
07571     VASSERT(thee != VNULL);
07572     VASSERT(thee->filled);
07573
07574     /* Get the mesh information */
07575     nx = thee->pmgp->nx;
07576     ny = thee->pmgp->ny;
07577     nz = thee->pmgp->nz;
07578     hx = thee->pmgp->hx;
07579     hy = thee->pmgp->hy;
07580     hzed = thee->pmgp->hzed;
07581     xmax = thee->xf[nx-1];
07582     ymax = thee->yf[ny-1];
07583     zmax = thee->zf[nz-1];
07584     xmin = thee->xf[0];
07585     ymin = thee->yf[0];
07586     zmin = thee->zf[0];
07587
07588     u = thee->u;
07589
07590     atom = Valist_getAtom(thee->pbe->alist, atomID);
07591
07592     /* Currently all atoms must be in the same partition. */
07593
07594     VASSERT(atom->partID != 0);
07595
07596     /* Convert the atom position to grid coordinates */
07597

```

```

07598     position = Vatom_getPosition(atom);
07599     ifloat = (position[0] - xmin)/hx;
07600     jfloat = (position[1] - ymin)/hy;
07601     kfloat = (position[2] - zmin)/hzd;
07602     ip1 = (int)ceil(ifloat);
07603     ip2 = ip1 + 2;
07604     im1 = (int)floor(ifloat);
07605     im2 = im1 - 2;
07606     jp1 = (int)ceil(jfloat);
07607     jp2 = jp1 + 2;
07608     jm1 = (int)floor(jfloat);
07609     jm2 = jm1 - 2;
07610     kp1 = (int)ceil(kfloat);
07611     kp2 = kp1 + 2;
07612     km1 = (int)floor(kfloat);
07613     km2 = km1 - 2;
07614
07615     /* This step shouldn't be necessary, but it saves nasty debugging
07616      * later on if something goes wrong */
07617     ip2 = VMIN2(ip2,nx-1);
07618     ip1 = VMIN2(ip1,nx-1);
07619     im1 = VMAX2(im1,0);
07620     im2 = VMAX2(im2,0);
07621     jp2 = VMIN2(jp2,ny-1);
07622     jp1 = VMIN2(jp1,ny-1);
07623     jm1 = VMAX2(jm1,0);
07624     jm2 = VMAX2(jm2,0);
07625     kp2 = VMIN2(kp2,nz-1);
07626     kp1 = VMIN2(kp1,nz-1);
07627     km1 = VMAX2(km1,0);
07628     km2 = VMAX2(km2,0);
07629
07630     /* Initialize observables to zero */
07631     energy = 0.0;
07632     pot = 0.0;
07633     for (i=0;i<3;i++) {
07634         rfe[i] = 0.0;
07635         for (j=0;j<3;j++) {
07636             rfde[i][j] = 0.0;
07637         }
07638     }
07639
07640     for (ii=im2; ii<=ip2; ii++) {
07641         mi = VFCHI4(ii,ifloat);
07642         mx = bspline4(mi);
07643         dmx = dbspline4(mi);
07644         d2mx = d2bspline4(mi);
07645         for (jj=jm2; jj<=jp2; jj++) {
07646             mj = VFCHI4(jj,jfloat);
07647             my = bspline4(mj);
07648             dmy = dbspline4(mj);
07649             d2my = d2bspline4(mj);
07650             for (kk=km2; kk<=kp2; kk++) {
07651                 mk = VFCHI4(kk,kfloat);
07652                 mz = bspline4(mk);
07653                 dmz = dbspline4(mk);
07654                 d2mz = d2bspline4(mk);

```

```

07655     f = u[IJK(ii,jj,kk)];
07656     /* potential */
07657     pot += f*mx*my*mz;
07658     /* field */
07659     rfe[0] += f*dmx*my*mz/hx;
07660     rfe[1] += f*mx*dmy*mz/hy;
07661     rfe[2] += f*mx*my*dmz/hzed;
07662     /* field gradient */
07663     rfde[0][0] += f*d2mx*my*mz/(hx*hx);
07664     rfde[1][0] += f*dmx*dmy*mz/(hy*hx);
07665     rfde[1][1] += f*mx*d2my*mz/(hy*hy);
07666     rfde[2][0] += f*dmx*my*dmz/(hx*hzed);
07667     rfde[2][1] += f*mx*dmy*dmz/(hy*hzed);
07668     rfde[2][2] += f*mx*my*d2mz/(hzed*hzed);
07669 }
07670 }
07671 }
07672 }
07673 charge = Vatom_getCharge(atom);
07674 dipole = Vatom_getDipole(atom);
07675 quad = Vatom_getQuadrupole(atom);
07676 qxx = quad[0]/3.0;
07677 qyx = quad[3]/3.0;
07678 qyy = quad[4]/3.0;
07679 qzx = quad[6]/3.0;
07680 qzy = quad[7]/3.0;
07681 qzz = quad[8]/3.0;
07682
07683 energy = pot * charge
07684     - rfe[0] * dipole[0]
07685     - rfe[1] * dipole[1]
07686     - rfe[2] * dipole[2]
07687     + rfde[0][0]*qxx
07688     + 2.0*rfde[1][0]*qyx + rfde[1][1]*qyy
07689     + 2.0*rfde[2][0]*qzx + 2.0*rfde[2][1]*qzy + rfde[2][2]*qzz;
07690
07691     return energy;
07692 }
07693
07694 VPUBLIC void Vpmg_fieldSpline4(Vpmg *thee, int atomID, double field[3]) {
07695
07696     Vatom *atom;
07697     double *u, f;
07698     /* Grid variables */
07699     int nx, ny, nz;
07700     double xmax, xmin, ymax, ymin, zmax, zmin;
07701     double hx, hy, hzed, ifloat, jfloat, kfloat;
07702     double *apos, position[3];
07703     /* B-Spline weights */
07704     double mx, my, mz, dmx, dmy, dmz;
07705     double mi, mj, mk;
07706     /* Loop indeces */
07707     int ip1, ip2, im1, im2, jp1, jp2, jm1, jm2, kp1, kp2, km1, km2;
07708     int i, j, ii, jj, kk;
07709
07710     VASSERT (thee != VNULL);

```

```

07712
07713     /* Get the mesh information */
07714     nx = thee->pmgp->nx;
07715     ny = thee->pmgp->ny;
07716     nz = thee->pmgp->nz;
07717     hx = thee->pmgp->hx;
07718     hy = thee->pmgp->hy;
07719     hzed = thee->pmgp->hzed;
07720     xmax = thee->xf[nx-1];
07721     ymax = thee->yf[ny-1];
07722     zmax = thee->zf[nz-1];
07723     xmin = thee->xf[0];
07724     ymin = thee->yf[0];
07725     zmin = thee->zf[0];
07726
07727     u = thee->u;
07728
07729     atom = Valist_getAtom(thee->pbe->alist, atomID);
07730
07731     /* Currently all atoms must be in the same partition. */
07732
07733     VASSERT (atom->partID != 0);
07734
07735     /* Convert the atom position to grid coordinates */
07736
07737     apos = Vatom_getPosition(atom);
07738     position[0] = apos[0] - xmin;
07739     position[1] = apos[1] - ymin;
07740     position[2] = apos[2] - zmin;
07741     ifloat = position[0]/hx;
07742     jfloat = position[1]/hy;
07743     kfloat = position[2]/hzed;
07744     ip1 = (int)ceil(ifloat);
07745     ip2 = ip1 + 2;
07746     im1 = (int)floor(ifloat);
07747     im2 = im1 - 2;
07748     jp1 = (int)ceil(jfloat);
07749     jp2 = jp1 + 2;
07750     jm1 = (int)floor(jfloat);
07751     jm2 = jm1 - 2;
07752     kp1 = (int)ceil(kfloat);
07753     kp2 = kp1 + 2;
07754     km1 = (int)floor(kfloat);
07755     km2 = km1 - 2;
07756
07757     /* This step shouldn't be necessary, but it saves nasty debugging
07758      * later on if something goes wrong */
07759     ip2 = VMIN2(ip2,nx-1);
07760     ip1 = VMIN2(ip1,nx-1);
07761     im1 = VMAX2(im1,0);
07762     im2 = VMAX2(im2,0);
07763     jp2 = VMIN2(jp2,ny-1);
07764     jp1 = VMIN2(jp1,ny-1);
07765     jm1 = VMAX2(jm1,0);
07766     jm2 = VMAX2(jm2,0);
07767     kp2 = VMIN2(kp2,nz-1);
07768     kp1 = VMIN2(kp1,nz-1);

```

```

07769     km1 = VMAX2(km1,0);
07770     km2 = VMAX2(km2,0);
07771
07772     for (i=0;i<3;i++) {
07773         field[i] = 0.0;
07774     }
07775
07776     for (ii=im2; ii<=ip2; ii++) {
07777         mi = VFCHI4(ii,ifloat);
07778         mx = bspline4(mi);
07779         dmx = dbspline4(mi);
07780         for (jj=jm2; jj<=jp2; jj++) {
07781             mj = VFCHI4(jj,jfloat);
07782             my = bspline4(mj);
07783             dmy = dbspline4(mj);
07784             for (kk=km2; kk<=kp2; kk++) {
07785                 mk = VFCHI4(kk,kfloat);
07786                 mz = bspline4(mk);
07787                 dmz = dbspline4(mk);
07788                 f = u[IJK(ii,jj,kk)];
07789
07790                 field[0] += f*dmx*my*mz/hx;
07791                 field[1] += f*mx*dmy*mz/hy;
07792                 field[2] += f*mx*my*dmz/hzed;
07793             }
07794         }
07795     }
07796 }
07797
07798 VPUBLIC void Vpmg_qfPermanentMultipoleForce(Vpmg *thee, int atomID,
07799                                         double force[3], double torque[3]) {
07800
07801     Vatom *atom;
07802     double f, *u, *apos, position[3];
07803
07804     /* Grid variables */
07805     int nx,ny,nz;
07806     double xlabel, ylabel, zlabel, xmin, ymin, zmin, xmax, ymax, zmax;
07807     double hx, hy, hzed, ifloat, jfloat, kfloat;
07808
07809     /* B-spline weights */
07810     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
07811     double mi, mj, mk;
07812
07813     /* Loop indeces */
07814     int i, j, k, ii, jj, kk;
07815     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07816
07817     /* Potential, field, field gradient and 2nd field gradient */
07818     double pot, e[3], de[3][3], d2e[3][3][3];
07819
07820     /* Permanent multipole components */
07821     double *dipole, *quad;
07822     double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
07823
07824     VASSERT(thee != VNNULL);
07825     VASSERT(thee->filled);

```

```

07826
07827     atom = Valist_getAtom(thee->pbe->alist, atomID);
07828
07829     /* Currently all atoms must be in the same partition. */
07830
07831     VASSERT(atom->partID != 0);
07832
07833     apos = VatomGetPosition(atom);
07834
07835     c = Vatom_getCharge(atom);
07836     dipole = Vatom_getDipole(atom);
07837     ux = dipole[0];
07838     uy = dipole[1];
07839     uz = dipole[2];
07840     quad = Vatom_getQuadrupole(atom);
07841     qxx = quad[0]/3.0;
07842     qxy = quad[1]/3.0;
07843     qxz = quad[2]/3.0;
07844     qyx = quad[3]/3.0;
07845     qyy = quad[4]/3.0;
07846     qyz = quad[5]/3.0;
07847     qzx = quad[6]/3.0;
07848     qzy = quad[7]/3.0;
07849     qzz = quad[8]/3.0;
07850
07851     /* Initialize observables */
07852     pot = 0.0;
07853     for (i=0;i<3;i++) {
07854         e[i] = 0.0;
07855         for (j=0;j<3;j++) {
07856             de[i][j] = 0.0;
07857             for (k=0;k<3;k++) {
07858                 d2e[i][j][k] = 0.0;
07859             }
07860         }
07861     }
07862
07863     /* Mesh info */
07864     nx = thee->pmgp->nx;
07865     ny = thee->pmgp->ny;
07866     nz = thee->pmgp->nz;
07867     hx = thee->pmgp->hx;
07868     hy = thee->pmgp->hy;
07869     hzed = thee->pmgp->hzed;
07870     xlen = thee->pmgp->xlen;
07871     ylen = thee->pmgp->ylen;
07872     zlen = thee->pmgp->zlen;
07873     xmin = thee->pmgp->xmin;
07874     ymin = thee->pmgp->ymin;
07875     zmin = thee->pmgp->zmin;
07876     xmax = thee->pmgp->xmax;
07877     ymax = thee->pmgp->ymax;
07878     zmax = thee->pmgp->zmax;
07879     u = thee->u;
07880
07881     /* Make sure we're on the grid */
07882     if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \

```

```

07883     || (apos[1]<=(ymin+2*hx)) || (apos[1]>=(ymax-2*hx)) \
07884     || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed)) {
07885         Vnm_print(2, "qfPermanentMultipoleForce: Atom off the mesh (ignoring) %6
.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
07886         fflush(stderr);
07887     } else {
07888
07889     /* Convert the atom position to grid coordinates */
07890     position[0] = apos[0] - xmin;
07891     position[1] = apos[1] - ymin;
07892     position[2] = apos[2] - zmin;
07893     ifloat = position[0]/hx;
07894     jfloat = position[1]/hy;
07895     kfloat = position[2]/hzed;
07896     ip1 = (int)ceil(ifloat);
07897     ip2 = ip1 + 2;
07898     im1 = (int)floor(ifloat);
07899     im2 = im1 - 2;
07900     jp1 = (int)ceil(jfloat);
07901     jp2 = jp1 + 2;
07902     jm1 = (int)floor(jfloat);
07903     jm2 = jm1 - 2;
07904     kp1 = (int)ceil(kfloat);
07905     kp2 = kp1 + 2;
07906     km1 = (int)floor(kfloat);
07907     km2 = km1 - 2;
07908
07909     /* This step shouldn't be necessary, but it saves nasty debugging
07910      * later on if something goes wrong */
07911     ip2 = VMIN2(ip2,nx-1);
07912     ip1 = VMIN2(ip1,nx-1);
07913     im1 = VMAX2(im1,0);
07914     im2 = VMAX2(im2,0);
07915     jp2 = VMIN2(jp2,ny-1);
07916     jp1 = VMIN2(jp1,ny-1);
07917     jm1 = VMAX2(jm1,0);
07918     jm2 = VMAX2(jm2,0);
07919     kp2 = VMIN2(kp2,nz-1);
07920     kp1 = VMIN2(kp1,nz-1);
07921     km1 = VMAX2(km1,0);
07922     km2 = VMAX2(km2,0);
07923
07924     for (ii=im2; ii<=ip2; ii++) {
07925         mi = VFCHI4(ii,ifloat);
07926         mx = bspline4(mi);
07927         dmx = dbspline4(mi);
07928         d2mx = d2bspline4(mi);
07929         d3mx = d3bspline4(mi);
07930         for (jj=jm2; jj<=jp2; jj++) {
07931             mj = VFCHI4(jj,jfloat);
07932             my = bspline4(mj);
07933             dmy = dbspline4(mj);
07934             d2my = d2bspline4(mj);
07935             d3my = d3bspline4(mj);
07936             for (kk=km2; kk<=kp2; kk++) {
07937                 mk = VFCHI4(kk,kfloat);
07938                 mz = bspline4(mk);

```

```

07939     dmz = dbspline4(mk);
07940     d2mz = d2bspline4(mk);
07941     d3mz = d3bspline4(mk);
07942     f = u[IJK(ii,jj,kk)];
07943     /* Potential */
07944     pot += f*mx*my*mz;
07945     /* Field */
07946     e[0] += f*dmx*my*mz/hx;
07947     e[1] += f*mx*dmy*mz/hy;
07948     e[2] += f*mx*my*dmz/hzed;
07949     /* Field gradient */
07950     de[0][0] += f*d2mx*my*mz/(hx*hx);
07951     de[1][0] += f*dmx*dmy*mz/(hy*hx);
07952     de[1][1] += f*mx*d2my*mz/(hy*hy);
07953     de[2][0] += f*dmx*my*dmz/(hx*hzed);
07954     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
07955     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
07956     /* 2nd Field Gradient
07957         VxVxVa */
07958     d2e[0][0][0] += f*d3mx*my*mz/(hx*hx*hx);
07959     d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
07960     d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
07961     /* VyVxVa */
07962     d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
07963     d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
07964     d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
07965     /* VyVyVa */
07966     d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
07967     d2e[1][1][1] += f*mx*d3my*mz/(hy*hy*hy);
07968     d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
07969     /* VzVxVa */
07970     d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
07971     d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
07972     d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
07973     /* VzVyVa */
07974     d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
07975     d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
07976     d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
07977     /* VzVzVa */
07978     d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
07979     d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
07980     d2e[2][2][2] += f*mx*my*d3mz/(hzed*hzed*hzed);
07981 }
07982 }
07983 }
07984 }
07985 }
07986 /* Monopole Force */
07987 force[0] = e[0]*c;
07988 force[1] = e[1]*c;
07989 force[2] = e[2]*c;
07990
07991 /* Dipole Force */
07992 force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
07993 force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
07994 force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
07995

```

```

07996     /* Quadrupole Force */
07997     force[0] += d2e[0][0][0]*qxx
07998         + d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
07999         + d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
08000     force[1] += d2e[0][0][1]*qxx
08001         + d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
08002         + d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
08003     force[2] += d2e[0][0][2]*qxx
08004         + d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
08005         + d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
08006
08007     /* Dipole Torque */
08008     torque[0] = uy * e[2] - uz * e[1];
08009     torque[1] = uz * e[0] - ux * e[2];
08010     torque[2] = ux * e[1] - uy * e[0];
08011     /* Quadrupole Torque */
08012     de[0][1] = de[1][0];
08013     de[0][2] = de[2][0];
08014     de[1][2] = de[2][1];
08015     torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08016                 - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08017     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08018                 - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08019     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08020                 - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08021
08022
08023     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08024         printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08025 }
08026
08027 VPUBLIC void Vpmg_ibPermanentMultipoleForce(Vpmg *thee, int atomID,
08028                                         double force[3]) {
08029
08030     Valist *alist;
08031     Vacc *acc;
08032     Vpbe *pbe;
08033     Vatom *atom;
08034     Vsurf_Meth srfm;
08035
08036     /* Grid variables */
08037     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
08038     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
08039     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
08040     double izmagic;
08041     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08042
08043     VASSERT(thee != VNULL);
08044
08045     /* Nonlinear PBE is not implemented for AMOEBA */
08046     VASSERT(!thee->pmgp->nonlin);
08047
08048     acc = thee->pbe->acc;
08049     srfm = thee->surfMeth;
08050     atom = Valist_getAtom(thee->pbe->alist, atomID);
08051
08052     /* Currently all atoms must be in the same partition. */

```

```

08053
08054     VASSERT(atom->partID != 0);
08055     apos = Vatom_getPosition(atom);
08056     arad = Vatom_getRadius(atom);
08057
08058     /* Reset force */
08059     force[0] = 0.0;
08060     force[1] = 0.0;
08061     force[2] = 0.0;
08062
08063     /* Get PBE info */
08064     pbe = thee->pbe;
08065     acc = pbe->acc;
08066     alist = pbe->alist;
08067     irad = Vpbe_getMaxIonRadius(pbe);
08068     zkappa2 = Vpbe_getZkappa2(pbe);
08069     izmagic = 1.0/Vpbe_getZmagic(pbe);
08070
08071     /* Should be a check for this further up. */
08072     VASSERT (zkappa2 > VPMGSMALL);
08073
08074     /* Mesh info */
08075     nx = thee->pmgp->nx;
08076     ny = thee->pmgp->ny;
08077     nz = thee->pmgp->nz;
08078     hx = thee->pmgp->hx;
08079     hy = thee->pmgp->hy;
08080     hzed = thee->pmgp->hzed;
08081     xlen = thee->pmgp->xlen;
08082     ylen = thee->pmgp->ylen;
08083     zlen = thee->pmgp->zlen;
08084     xmin = thee->pmgp->xmin;
08085     ymin = thee->pmgp->ymin;
08086     zmin = thee->pmgp->zmin;
08087     xmax = thee->pmgp->xmax;
08088     ymax = thee->pmgp->ymax;
08089     zmax = thee->pmgp->zmax;
08090
08091     /* Make sure we're on the grid */
08092     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
08093         (apos[1]<=ymin) || (apos[1]>=ymax) || \
08094         (apos[2]<=zmin) || (apos[2]>=zmax)) {
08095         Vnm_print(2, "ibPermanentMultipoleForce: Atom %d at (%4.3f, %4.3f, %4.3f
08096             ) is off the mesh (ignoring):\n", atomID, apos[0], apos[1], apos[2]);
08097         Vnm_print(2, "ibPermanentMultipoleForce: xmin = %g, xmax = %g\n", xmin,
08098             xmax);
08099         Vnm_print(2, "ibPermanentMultipoleForce: ymin = %g, ymax = %g\n", ymin,
08100             ymax);
08101         Vnm_print(2, "ibPermanentMultipoleForce: zmin = %g, zmax = %g\n", zmin,
08102             zmax);
08103         fflush(stderr);
08104     } else {
08105
08106         /* Convert the atom position to grid reference frame */
08107         position[0] = apos[0] - xmin;
08108         position[1] = apos[1] - ymin;
08109         position[2] = apos[2] - zmin;

```

```

08106     /* Integrate over points within this atom's (inflated) radius */
08107     rtot = (irad + arad + theee->splineWin);
08108     rtot2 = VSQR(rtot);
08109     dx = rtot + 0.5*hx;
08110     imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
08111     imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
08112     for (i=imin; i<=imax; i++) {
08113         dx2 = VSQR(position[0] - hx*i);
08114         if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
08115         else dy = 0.5*hy;
08116         jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
08117         jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
08118         for (j=jmin; j<=jmax; j++) {
08119             dy2 = VSQR(position[1] - hy*j);
08120             if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
08121             else dz = 0.5*hzed;
08122             kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
08123             kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
08124             for (k=kmin; k<=kmax; k++) {
08125                 dz2 = VSQR(k*hzed - position[2]);
08126                 /* See if grid point is inside ivdw radius and set ccf
08127                  * accordingly (do spline assignment here) */
08128                 if ((dz2 + dy2 + dx2) <= rtot2) {
08129                     gpos[0] = i*hx + xmin;
08130                     gpos[1] = j*hy + ymin;
08131                     gpos[2] = k*hzed + zmin;
08132                     Vpmg_splineSelect(srfm, acc, gpos, theee->splineWin, irad,
08133                         atom, tgrad);
08134                     fmag = VSQR(theee->u[IJK(i,j,k)])*theee->kappa[IJK(i,j,k)];
08135                     force[0] += (zkappa2*fmag*tgrad[0]);
08136                     force[1] += (zkappa2*fmag*tgrad[1]);
08137                     force[2] += (zkappa2*fmag*tgrad[2]);
08138                 }
08139             } /* k loop */
08140         } /* j loop */
08141     } /* i loop */
08142 }
08143
08144     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
08145     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
08146     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
08147
08148 }
08149
08150 VPUBLIC void Vpmg_dbPermanentMultipoleForce(Vpmg *thee, int atomID,
08151                                         double force[3]) {
08152
08153     Vacc *acc;
08154     Vpbe *pbe;
08155     Vatom *atom;
08156     Vsurf_Meth srfm;
08157
08158     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
08159     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
08160     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;

```

```

08161     double *u, Hxijk, Hyijk, Hziijk, Hxim1jk, Hyijm1k, Hzijkml;
08162     double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
08163     double dHzijkml[3];
08164     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08165
08166     VASSERT(thee != VNULL);
08167
08168     acc = thee->pbe->acc;
08169     srfm = thee->surfMeth;
08170     atom = Valist_getAtom(thee->pbe->alist, atomID);
08171
08172     /* Currently all atoms must be in the same partition. */
08173
08174     VASSERT(atom->partID != 0);
08175     arad = Vatom_getRadius(atom);
08176     apos = Vatom_getPosition(atom);
08177
08178     /* Reset force */
08179     force[0] = 0.0;
08180     force[1] = 0.0;
08181     force[2] = 0.0;
08182
08183     /* Get PBE info */
08184     pbe = thee->pbe;
08185     acc = pbe->acc;
08186     epsp = Vpbe_getSoluteDiel(pbe);
08187     epsw = Vpbe_getSolventDiel(pbe);
08188     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
08189     izmagic = 1.0/Vpbe_getZmagic(pbe);
08190
08191
08192     deps = (epsw - epsp);
08193     depsi = 1.0/deps;
08194
08195     VASSERT(VABS(deps) > VPMGSMALL);
08196
08197     /* Mesh info */
08198     nx = thee->pmgp->nx;
08199     ny = thee->pmgp->ny;
08200     nz = thee->pmgp->nz;
08201     hx = thee->pmgp->hx;
08202     hy = thee->pmgp->hy;
08203     hzed = thee->pmgp->hzed;
08204     xlen = thee->pmgp->xlen;
08205     ylen = thee->pmgp->ylen;
08206     zlen = thee->pmgp->zlen;
08207     xmin = thee->pmgp->xmin;
08208     ymin = thee->pmgp->ymin;
08209     zmin = thee->pmgp->zmin;
08210     xmax = thee->pmgp->xmax;
08211     ymax = thee->pmgp->ymax;
08212     zmax = thee->pmgp->zmax;
08213     u = thee->u;
08214
08215     /* Make sure we're on the grid */
08216     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
08217         (apos[1]<=ymin) || (apos[1]>=ymax) || \

```

```

08218     (apos[2]<=zmin) || (apos[2]>=zmax)) {
08219         Vnm_print(2, "dbPermanentMultipoleForce: Atom at (%4.3f, %4.3f, %4.3f) i
08220             s off the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
08221         Vnm_print(2, "dbPermanentMultipoleForce: xmin = %g, xmax = %g\n", xmin,
08222             xmax);
08221         Vnm_print(2, "dbPermanentMultipoleForce: ymin = %g, ymax = %g\n", ymin,
08222             ymax);
08222         Vnm_print(2, "dbPermanentMultipoleForce: zmin = %g, zmax = %g\n", zmin,
08223             zmax);
08223         fflush(stderr);
08224     } else {
08225
08226         /* Convert the atom position to grid reference frame */
08227         position[0] = apos[0] - xmin;
08228         position[1] = apos[1] - ymin;
08229         position[2] = apos[2] - zmin;
08230
08231         /* Integrate over points within this atom's (inflated) radius */
08232         rtot = (arad + thee->splineWin);
08233         rtot2 = VSQR(rtot);
08234         dx = rtot/hx;
08235         imin = (int)floor((position[0]-rtot)/hx);
08236         if (imin < 1) {
08237             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08238             return;
08239         }
08240         imax = (int)ceil((position[0]+rtot)/hx);
08241         if (imax > (nx-2)) {
08242             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08243             return;
08244         }
08245         jmin = (int)floor((position[1]-rtot)/hy);
08246         if (jmin < 1) {
08247             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08248             return;
08249         }
08250         jmax = (int)ceil((position[1]+rtot)/hy);
08251         if (jmax > (ny-2)) {
08252             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08253             return;
08254         }
08255         kmin = (int)floor((position[2]-rtot)/hzed);
08256         if (kmin < 1) {
08257             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08258             return;
08259         }
08260         kmax = (int)ceil((position[2]+rtot)/hzed);
08261         if (kmax > (nz-2)) {
08262             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08263             return;
08264         }
08265         for (i=imin; i<=imax; i++) {
08266             for (j=jmin; j<=jmax; j++) {
08267                 for (k=kmin; k<=kmax; k++) {
08268                     /* i,j,k */
08269                     gpos[0] = (i+0.5)*hx + xmin;
08270                     gpos[1] = j*hy + ymin;

```

```

08271     gpos[2] = k*hzed + zmin;
08272     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
08273     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08274         atom, dHxijk);
08275     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
08276     gpos[0] = i*hx + xmin;
08277     gpos[1] = (j+0.5)*hy + ymin;
08278     gpos[2] = k*hzed + zmin;
08279     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
08280     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08281         atom, dHyijk);
08282     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
08283     gpos[0] = i*hx + xmin;
08284     gpos[1] = j*hy + ymin;
08285     gpos[2] = (k+0.5)*hzed + zmin;
08286     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
08287     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08288         atom, dHzijk);
08289     for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
08290     /* i-1, j, k */
08291     gpos[0] = (i-0.5)*hx + xmin;
08292     gpos[1] = j*hy + ymin;
08293     gpos[2] = k*hzed + zmin;
08294     Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
08295     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08296         atom, dHxim1jk);
08297     for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
08298     /* i, j-1, k */
08299     gpos[0] = i*hx + xmin;
08300     gpos[1] = (j-0.5)*hy + ymin;
08301     gpos[2] = k*hzed + zmin;
08302     Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
08303     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08304         atom, dHyijm1k);
08305     for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
08306     /* i, j, k-1 */
08307     gpos[0] = i*hx + xmin;
08308     gpos[1] = j*hy + ymin;
08309     gpos[2] = (k-0.5)*hzed + zmin;
08310     Hzijkml = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
08311     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08312         atom, dHzijkml);
08313     for (l=0; l<3; l++) dHzijkml[l] *= Hzijkml;
08314     dbFmag = u[IJK(i,j,k)];
08315     tgrad[0] =
08316         (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08317         + dHxim1jk[0]* (u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
08318         + (dHyijk[0] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
08319         + dHyijm1k[0]* (u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
08320         + (dHzijk[0] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
08321         + dHzijkml[0]* (u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
08322     tgrad[1] =
08323         (dHxijk[1] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08324         + dHxim1jk[1]* (u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
08325         + (dHyijk[1] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
08326         + dHyijm1k[1]* (u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
08327         + (dHzijk[1] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))

```

```

08328     + dHzijkml[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
08329     tgrad[2] =
08330         (dHxijk[2]* (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08331         + dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
08332         + (dHyijk[2]* (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
08333         + dHyjm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
08334         + (dHzijk[2]* (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
08335         + dHzijkml[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
08336     force[0] += (dbFmag*tgrad[0]);
08337     force[1] += (dbFmag*tgrad[1]);
08338     force[2] += (dbFmag*tgrad[2]);
08339     } /* k loop */
08340 } /* j loop */
08341 } /* i loop */
08342 force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
08343 force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
08344 force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
08345 }
08346 }
08347
08348 VPUBLIC void Vpmg_qfDirectPolForce (Vpmg *thee, Vgrid* perm, Vgrid *induced,
08349                                         int atomID, double force[3], double torque[3])
08350 {
08351     Vatom *atom;
08352     Vpbe *pbe;
08353     double f, fp, *u, *up, *apos, position[3];
08354
08355     /* Grid variables */
08356     int nx,ny,nz;
08357     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08358     double hx, hy, hzed, ifloat, jfloat, kfloat;
08359
08360     /* B-spline weights */
08361     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08362     double mi, mj, mk;
08363
08364     /* Loop indeces */
08365     int i, j, k, ii, jj, kk;
08366     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
08367
08368     /* Permanent potential, field, field gradient and 2nd field gradient */
08369     double pot, e[3], de[3][3], d2e[3][3][3];
08370     /* Induced dipole field */
08371     double dep[3][3];
08372
08373     /* Permanent multipole components */
08374     double *dipole, *quad;
08375     double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08376     double uix, uiy, uiz;
08377
08378     VASSERT(thee != VNULL);
08379     VASSERT(induced != VNULL); /* the potential due to permanent multipoles.*/
08380     VASSERT(induced != VNULL); /* the potential due to local induced dipoles.*/
08381     VASSERT(thee->pbe != VNULL);
08382     VASSERT(thee->pbe->alist != VNULL);
08383

```

```

08384     atom = Valist_getAtom(thee->pbe->alist, atomID);
08385     VASSERT(atom->partID != 0); /* all atoms must be in the same partition.*/
08386     apos = Vatom_getPosition(atom);
08387
08388     c = Vatom_getCharge(atom);
08389     dipole = Vatom_getDipole(atom);
08390     ux = dipole[0];
08391     uy = dipole[1];
08392     uz = dipole[2];
08393     quad = Vatom_getQuadrupole(atom);
08394     qxx = quad[0]/3.0;
08395     qxy = quad[1]/3.0;
08396     qxz = quad[2]/3.0;
08397     qyx = quad[3]/3.0;
08398     qyy = quad[4]/3.0;
08399     qyz = quad[5]/3.0;
08400     qzx = quad[6]/3.0;
08401     qzy = quad[7]/3.0;
08402     qzz = quad[8]/3.0;
08403
08404     dipole = Vatom_getInducedDipole(atom);
08405     uix = dipole[0];
08406     uiy = dipole[1];
08407     uiz = dipole[2];
08408
08409     /* Reset Field Gradients */
08410     pot = 0.0;
08411     for (i=0;i<3;i++) {
08412         e[i] = 0.0;
08413         for (j=0;j<3;j++) {
08414             de[i][j] = 0.0;
08415             dep[i][j] = 0.0;
08416             for (k=0;k<3;k++) {
08417                 d2e[i][j][k] = 0.0;
08418             }
08419         }
08420     }
08421
08422     /* Mesh info */
08423     nx = thee->pmgp->nx;
08424     ny = thee->pmgp->ny;
08425     nz = thee->pmgp->nz;
08426     hx = thee->pmgp->hx;
08427     hy = thee->pmgp->hy;
08428     hzed = thee->pmgp->hzed;
08429     xlen = thee->pmgp->xlen;
08430     ylen = thee->pmgp->ylen;
08431     zlen = thee->pmgp->zlen;
08432     xmin = thee->pmgp->xmin;
08433     ymin = thee->pmgp->ymin;
08434     zmin = thee->pmgp->zmin;
08435     xmax = thee->pmgp->xmax;
08436     ymax = thee->pmgp->ymax;
08437     zmax = thee->pmgp->zmax;
08438     u = induced->data;
08439     up = perm->data;
08440

```

```

08441     /* Make sure we're on the grid */
08442     if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
08443         || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
08444         || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08445         Vnm_print(2, "qfDirectPolForce: Atom off the mesh (ignoring) %6.3f %6.3f
08446             %6.3f\n", apos[0], apos[1], apos[2]);
08447         fflush(stderr);
08448     } else {
08449         /* Convert the atom position to grid coordinates */
08450         position[0] = apos[0] - xmin;
08451         position[1] = apos[1] - ymin;
08452         position[2] = apos[2] - zmin;
08453         ifloat = position[0]/hx;
08454         jfloat = position[1]/hy;
08455         kfloat = position[2]/hzed;
08456         ip1 = (int)ceil(ifloat);
08457         ip2 = ip1 + 2;
08458         im1 = (int)floor(ifloat);
08459         im2 = im1 - 2;
08460         jp1 = (int)ceil(jfloat);
08461         jp2 = jp1 + 2;
08462         jm1 = (int)floor(jfloat);
08463         jm2 = jm1 - 2;
08464         kp1 = (int)ceil(kfloat);
08465         kp2 = kp1 + 2;
08466         km1 = (int)floor(kfloat);
08467         km2 = km1 - 2;
08468
08469         /* This step shouldn't be necessary, but it saves nasty debugging
08470            * later on if something goes wrong */
08471         ip2 = VMIN2(ip2,nx-1);
08472         ip1 = VMIN2(ip1,nx-1);
08473         im1 = VMAX2(im1,0);
08474         im2 = VMAX2(im2,0);
08475         jp2 = VMIN2(jp2,ny-1);
08476         jp1 = VMIN2(jp1,ny-1);
08477         jm1 = VMAX2(jm1,0);
08478         jm2 = VMAX2(jm2,0);
08479         kp2 = VMIN2(kp2,nz-1);
08480         kp1 = VMIN2(kp1,nz-1);
08481         km1 = VMAX2(km1,0);
08482         km2 = VMAX2(km2,0);
08483
08484         for (ii=im2; ii<=ip2; ii++) {
08485             mi = VFCHI4(ii,ifloat);
08486             mx = bspline4(mi);
08487             dmx = dbspline4(mi);
08488             d2mx = d2bspline4(mi);
08489             d3mx = d3bspline4(mi);
08490             for (jj=jm2; jj<=jp2; jj++) {
08491                 mj = VFCHI4(jj,jfloat);
08492                 my = bspline4(mj);
08493                 dmy = dbspline4(mj);
08494                 d2my = d2bspline4(mj);
08495                 d3my = d3bspline4(mj);
08496

```

```

08497   for (kk=km2; kk<=kp2; kk++) {
08498     mk = VFCHI4(kk,kfloat);
08499     mz = bspline4(mk);
08500     dmz = dbspline4(mk);
08501     d2mz = d2bspline4(mk);
08502     d3mz = d3bspline4(mk);
08503     f = u[IJK(ij,jj,kk)];
08504     fp = up[IJK(ij,jj,kk)];
08505     /* The potential */
08506     pot += f*mx*my*mz;
08507     /* The field */
08508     e[0] += f*dmx*my*mz/hx;
08509     e[1] += f*mx*dmy*mz/hy;
08510     e[2] += f*mx*my*dmz/hzed;
08511     /* The gradient of the field */
08512     de[0][0] += f*d2mx*my*mz/(hx*hx);
08513     de[1][0] += f*dmx*cmy*mz/(hy*hy);
08514     de[1][1] += f*mx*d2my*mz/(hy*hy);
08515     de[2][0] += f*dmx*my*dmz/(hx*hzed);
08516     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
08517     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
08518     /* The gradient of the (permanent) field */
08519     dep[0][0] += fp*d2mx*my*mz/(hx*hx);
08520     dep[1][0] += fp*dmx*dmy*mz/(hy*hy);
08521     dep[1][1] += fp*mx*d2my*mz/(hy*hy);
08522     dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
08523     dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
08524     dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
08525     /* The 2nd gradient of the field
08526      VxVxVa */
08527     d2e[0][0][0] += f*d3mx*my*mz/(hx*hx*hx);
08528     d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
08529     d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
08530     /* VyVxVa */
08531     d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
08532     d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
08533     d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
08534     /* VyVyVa */
08535     d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
08536     d2e[1][1][1] += f*mx*d3my*mz/(hy*hy*hy);
08537     d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
08538     /* VzVxVa */
08539     d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
08540     d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
08541     d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
08542     /* VzVyVa */
08543     d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
08544     d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
08545     d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08546     /* VzVzVa */
08547     d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
08548     d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08549     d2e[2][2][2] += f*mx*my*d3mz/(hzed*hzed*hzed);
08550   }
08551 }
08552 }
08553 }
```

```

08554
08555     /* force on permanent multipole due to induced reaction field */
08556
08557     /* Monopole Force */
08558     force[0] = e[0]*c;
08559     force[1] = e[1]*c;
08560     force[2] = e[2]*c;
08561
08562     /* Dipole Force */
08563     force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08564     force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08565     force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08566
08567     /* Quadrupole Force */
08568     force[0] += d2e[0][0]*qxx
08569         + d2e[1][0]*qyx*2.0+d2e[1][1]*qyy
08570         + d2e[2][0]*qzx*2.0+d2e[2][1]*qzy*2.0+d2e[2][2]*qzz;
08571     force[1] += d2e[0][1]*qxx
08572         + d2e[1][1]*qyx*2.0+d2e[1][2]*qyy
08573         + d2e[2][1]*qzx*2.0+d2e[2][2]*qzy*2.0+d2e[2][3]*qzz;
08574     force[2] += d2e[0][2]*qxx
08575         + d2e[1][2]*qyx*2.0+d2e[1][3]*qyy
08576         + d2e[2][2]*qzx*2.0+d2e[2][3]*qzy*2.0+d2e[2][4]*qzz;
08577
08578     /* torque on permanent mulitpole due to induced reaction field */
08579
08580     /* Dipole Torque */
08581     torque[0] = uy * e[2] - uz * e[1];
08582     torque[1] = uz * e[0] - ux * e[2];
08583     torque[2] = ux * e[1] - uy * e[0];
08584
08585     /* Quadrupole Torque */
08586     /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
08587     Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
08588     Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx)) */
08589     de[0][1] = de[1][0];
08590     de[0][2] = de[2][0];
08591     de[1][2] = de[2][1];
08592     torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08593                     - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08594     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08595                     - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08596     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08597                     - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08598
08599     /* force on induced dipole due to permanent reaction field */
08600
08601     force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
08602     force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
08603     force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
08604
08605     force[0] = 0.5 * force[0];
08606     force[1] = 0.5 * force[1];
08607     force[2] = 0.5 * force[2];
08608     torque[0] = 0.5 * torque[0];
08609     torque[1] = 0.5 * torque[1];
08610     torque[2] = 0.5 * torque[2];

```

```

08611
08612     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08613         printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08614 }
08615
08616 VPUBLIC void Vpmg_qfNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
08617                                     int atomID, double force[3], double torque[3]
08618 ) {
08619     Vatom *atom;
08620     double *apos, *dipole, *quad, position[3], hx, hy, hzed;
08621     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08622     double pot, e[3], de[3][3], dep[3][3], d2e[3][3][3];
08623     double mx, my, mz, dmx, dmy, dmz, mi, mj, mk;
08624     double d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08625     double *u, *up, charge, ifloat, jfloat, kfloat;
08626     double f, fp, c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08627     double uix, uiy, uiz;
08628     int i, j, k, nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
08629     int kp1, kp2, ii, jj, kk;
08630
08631     VASSERT(thee != VNULL);
08632     VASSERT(perm != VNULL); /* potential due to permanent multipoles. */
08633     VASSERT(nlInduced != VNULL); /* potential due to non-local induced dipoles */
08634
08635     VASSERT(!thee->pmpg->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */
08636
08637     atom = Valist_getAtom(thee->pbe->alist, atomID);
08638     VASSERT(atom->partID != 0); /* Currently all atoms must be in the same part
08639     ition. */
08640     apos = VatomGetPosition(atom);
08641     c = Vatom_getCharge(atom);
08642     dipole = Vatom_getDipole(atom);
08643     ux = dipole[0];
08644     uy = dipole[1];
08645     uz = dipole[2];
08646     quad = Vatom_getQuadrupole(atom);
08647     qxx = quad[0]/3.0;
08648     qxy = quad[1]/3.0;
08649     qxz = quad[2]/3.0;
08650     qyx = quad[3]/3.0;
08651     qyy = quad[4]/3.0;
08652     qyz = quad[5]/3.0;
08653     qzx = quad[6]/3.0;
08654     qzy = quad[7]/3.0;
08655     qzz = quad[8]/3.0;
08656
08657     dipole = Vatom_getNLInducedDipole(atom);
08658     uix = dipole[0];
08659     uiy = dipole[1];
08660     uiz = dipole[2];
08661
08662     /* Reset Field Gradients */
08663     pot = 0.0;
08664     for (i=0; i<3; i++) {

```

```

08664     e[i] = 0.0;
08665     for (j=0;j<3;j++) {
08666         de[i][j] = 0.0;
08667         dep[i][j] = 0.0;
08668         for (k=0;k<3;k++) {
08669             d2e[i][j][k] = 0.0;
08670         }
08671     }
08672 }
08673
08674 /* Mesh info */
08675 nx = thee->pmgp->nx;
08676 ny = thee->pmgp->ny;
08677 nz = thee->pmgp->nz;
08678 hx = thee->pmgp->hx;
08679 hy = thee->pmgp->hy;
08680 hzed = thee->pmgp->hzed;
08681 xlen = thee->pmgp->xlen;
08682 ylen = thee->pmgp->ylen;
08683 zlen = thee->pmgp->zlen;
08684 xmin = thee->pmgp->xmin;
08685 ymin = thee->pmgp->ymin;
08686 zmin = thee->pmgp->zmin;
08687 xmax = thee->pmgp->xmax;
08688 ymax = thee->pmgp->ymax;
08689 zmax = thee->pmgp->zmax;
08690 u = nlInduced->data;
08691 up = perm->data;
08692
08693
08694 /* Make sure we're on the grid */
08695 if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
08696 || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
08697 || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08698     Vnm_print(2, "qfNLDirectMultipoleForce: Atom off the mesh (ignoring) %.
3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
08699 } else {
08700
08701     /* Convert the atom position to grid coordinates */
08702     position[0] = apos[0] - xmin;
08703     position[1] = apos[1] - ymin;
08704     position[2] = apos[2] - zmin;
08705     ifloat = position[0]/hx;
08706     jfloat = position[1]/hy;
08707     kfloat = position[2]/hzed;
08708     ip1 = (int)ceil(ifloat);
08709     ip2 = ip1 + 2;
08710     im1 = (int)floor(ifloat);
08711     im2 = im1 - 2;
08712     jp1 = (int)ceil(jfloat);
08713     jp2 = jp1 + 2;
08714     jm1 = (int)floor(jfloat);
08715     jm2 = jm1 - 2;
08716     kp1 = (int)ceil(kfloat);
08717     kp2 = kp1 + 2;
08718     km1 = (int)floor(kfloat);
08719     km2 = km1 - 2;

```

```

08720
08721     /* This step shouldn't be necessary, but it saves nasty debugging
08722     * later on if something goes wrong */
08723     ip2 = VMIN2(ip2,nx-1);
08724     ip1 = VMIN2(ip1,nx-1);
08725     im1 = VMAX2(im1,0);
08726     im2 = VMAX2(im2,0);
08727     jp2 = VMIN2(jp2,ny-1);
08728     jp1 = VMIN2(jp1,ny-1);
08729     jm1 = VMAX2(jm1,0);
08730     jm2 = VMAX2(jm2,0);
08731     kp2 = VMIN2(kp2,nz-1);
08732     kp1 = VMIN2(kp1,nz-1);
08733     km1 = VMAX2(km1,0);
08734     km2 = VMAX2(km2,0);
08735
08736     for (ii=im2; ii<=ip2; ii++) {
08737         mi = VFCHI4(ii,ifloat);
08738         mx = bspline4(mi);
08739         dmx = dbspline4(mi);
08740         d2mx = d2bspline4(mi);
08741         d3mx = d3bspline4(mi);
08742         for (jj=jm2; jj<=jp2; jj++) {
08743             mj = VFCHI4(jj,jfloat);
08744             my = bspline4(mj);
08745             dmy = dbspline4(mj);
08746             d2my = d2bspline4(mj);
08747             d3my = d3bspline4(mj);
08748             for (kk=km2; kk<=kp2; kk++) {
08749                 mk = VFCHI4(kk,kfloat);
08750                 mz = bspline4(mk);
08751                 dmz = dbspline4(mk);
08752                 d2mz = d2bspline4(mk);
08753                 d3mz = d3bspline4(mk);
08754                 f = u[IJK(ii,jj,kk)];
08755                 fp = up[IJK(ii,jj,kk)];
08756                 /* The potential */
08757                 pot += f*mx*my*mz;
08758                 /* The field */
08759                 e[0] += f*dmx*my*mz/hx;
08760                 e[1] += f*mx*dmy*mz/hy;
08761                 e[2] += f*mx*my*dmz/hzed;
08762                 /* The gradient of the field */
08763                 de[0][0] += f*d2mx*my*mz/(hx*hx);
08764                 de[1][0] += f*dmx*dmy*mz/(hy*hx);
08765                 de[1][1] += f*mx*d2my*mz/(hy*hy);
08766                 de[2][0] += f*dmx*my*dmz/(hx*hzed);
08767                 de[2][1] += f*mx*dmy*dmz/(hy*hzed);
08768                 de[2][2] += f*mx*my*d2mz/(hzed*hzed);
08769                 /* The gradient of the (permanent) field */
08770                 dep[0][0] += fp*d2mx*my*mz/(hx*hx);
08771                 dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
08772                 dep[1][1] += fp*mx*d2my*mz/(hy*hy);
08773                 dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
08774                 dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
08775                 dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
08776                 /* The 2nd gradient of the field */

```

```

08777     /* VxVxVa */
08778     d2e[0][0][0] += f*d3mx*my*mz / (hx*hx*hx);
08779     d2e[0][0][1] += f*d2mx*dmy*mz / (hx*hy*hx);
08780     d2e[0][0][2] += f*d2mx*my*dmz / (hx*hx*hzed);
08781     /* VyVxVa */
08782     d2e[1][0][0] += f*d2mx*dmy*mz / (hx*hx*hy);
08783     d2e[1][0][1] += f*dmx*d2my*mz / (hx*hy*hy);
08784     d2e[1][0][2] += f*dmx*dmy*dmz / (hx*hy*hzed);
08785     /* VyVyVa */
08786     d2e[1][1][0] += f*dmx*d2my*mz / (hx*hy*hy);
08787     d2e[1][1][1] += f*mx*d3my*mz / (hy*hy*hy);
08788     d2e[1][1][2] += f*mx*d2my*dmz / (hy*hy*hzed);
08789     /* VzVxVa */
08790     d2e[2][0][0] += f*d2mx*my*dmz / (hx*hx*hzed);
08791     d2e[2][0][1] += f*dmx*dmy*dmz / (hx*hy*hzed);
08792     d2e[2][0][2] += f*dmx*my*d2mz / (hx*hzed*hzed);
08793     /* VzVyVa */
08794     d2e[2][1][0] += f*dmx*dmy*dmz / (hx*hy*hzed);
08795     d2e[2][1][1] += f*mx*d2my*dmz / (hy*hy*hzed);
08796     d2e[2][1][2] += f*mx*dmy*d2mz / (hy*hzed*hzed);
08797     /* VzVzVa */
08798     d2e[2][2][0] += f*dmx*my*d2mz / (hx*hzed*hzed);
08799     d2e[2][2][1] += f*mx*dmy*d2mz / (hy*hzed*hzed);
08800     d2e[2][2][2] += f*mx*my*d3mz / (hzed*hzed*hzed);
08801 }
08802 }
08803 }
08804 }
08805
08806 /* force on permanent multipole due to non-local induced reaction field */
08807
08808 /* Monopole Force */
08809 force[0] = e[0]*c;
08810 force[1] = e[1]*c;
08811 force[2] = e[2]*c;
08812
08813 /* Dipole Force */
08814 force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08815 force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08816 force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08817
08818 /* Quadrupole Force */
08819 force[0] += d2e[0][0][0]*qxx
08820     + d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
08821     + d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
08822 force[1] += d2e[0][0][1]*qxx
08823     + d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
08824     + d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
08825 force[2] += d2e[0][0][2]*qxx
08826     + d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
08827     + d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
08828
08829 /* torque on permanent mulitpole due to non-local induced reaction field */
08830
08831 /* Dipole Torque */
08832 torque[0] = uy * e[2] - uz * e[1];
08833 torque[1] = uz * e[0] - ux * e[2];

```

```

08834     torque[2] = ux * e[1] - uy * e[0];
08835
08836     /* Quadrupole Torque */
08837     /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby)) */
08838     Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz));
08839     Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx)); */
08840     de[0][1] = de[1][0];
08841     de[0][2] = de[2][0];
08842     de[1][2] = de[2][1];
08843     torque[0] = 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08844           - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08845     torque[1] = 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08846           - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08847     torque[2] = 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08848           - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08849
08850     /* force on non-local induced dipole due to permanent reaction field */
08851
08852     force[0] = dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
08853     force[1] = dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
08854     force[2] = dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
08855
08856     force[0] = 0.5 * force[0];
08857     force[1] = 0.5 * force[1];
08858     force[2] = 0.5 * force[2];
08859     torque[0] = 0.5 * torque[0];
08860     torque[1] = 0.5 * torque[1];
08861     torque[2] = 0.5 * torque[2];
08862
08863     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08864      printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08865 }
08866
08867 VPUBLIC void Vpmg_ibDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *induced,
08868                               int atomID, double force[3]) {
08869
08870     Vatom *atom;
08871     Valist *alist;
08872     Vacc *acc;
08873     Vpbe *pbe;
08874     Vsurf_Meth srfm;
08875
08876     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
08877     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
08878     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
08879     double izmagic;
08880     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08881
08882     VASSERT(thee != VNULL);
08883     VASSERT(perm != VNULL); /* potential due to permanent multipoles.*/
08884     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
08885     VASSERT (!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA
08886 */
08887     acc = thee->pbe->acc;
08888     srfm = thee->surfMeth;
08889     atom = Valist_getAtom(thee->pbe->alist, atomID);

```

```

08890     VASSERT(atom->partID != 0); /* Currently all atoms must be in the same part
08891     ition. */
08892     apos = Vatom_getPosition(atom);
08893     arad = Vatom_getRadius(atom);
08894     /* Reset force */
08895     force[0] = 0.0;
08896     force[1] = 0.0;
08897     force[2] = 0.0;
08898
08899     /* Get PBE info */
08900     pbe = thee->pbe;
08901     acc = pbe->acc;
08902     alist = pbe->alist;
08903     irad = Vpbe_getMaxIonRadius(pbe);
08904     zkappa2 = Vpbe_getZkappa2(pbe);
08905     izmagic = 1.0/Vpbe_getZmagic(pbe);
08906
08907     VASSERT (zkappa2 > VPMGSMALL); /* It is ok to run AMOEBA with no ions, but th
08908     is is checked for higher up in the driver. */
08909
08910     /* Mesh info */
08911     nx = induced->nx;
08912     ny = induced->ny;
08913     nz = induced->nz;
08914     hx = induced->hx;
08915     hy = induced->hy;
08916     hzed = induced->hzed;
08917     xmin = induced->xmin;
08918     ymin = induced->ymin;
08919     zmin = induced->zmin;
08920     xmax = induced->xmax;
08921     ymax = induced->ymax;
08922     zmax = induced->zmax;
08923     xlen = xmax-xmin;
08924     ylen = ymax-ymin;
08925     zlen = zmax-zmin;
08926
08927     /* Make sure we're on the grid */
08928     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
08929         (apos[1]<=ymin) || (apos[1]>=ymax) || \
08930         (apos[2]<=zmin) || (apos[2]>=zmax)) {
08931         Vnm_print(2, "Vpmg_ibForce: Atom at (%4.3f, %4.3f, %4.3f) is off the mes
08932         h (ignoring):\n",
08933             apos[0], apos[1], apos[2]);
08934         Vnm_print(2, "Vpmg_ibForce: xmin = %g, xmax = %g\n", xmin, xmax);
08935         Vnm_print(2, "Vpmg_ibForce: ymin = %g, ymax = %g\n", ymin, ymax);
08936         Vnm_print(2, "Vpmg_ibForce: zmin = %g, zmax = %g\n", zmin, zmax);
08937         fflush(stderr);
08938     } else {
08939
08940         /* Convert the atom position to grid reference frame */
08941         position[0] = apos[0] - xmin;
08942         position[1] = apos[1] - ymin;
08943         position[2] = apos[2] - zmin;
08944
08945         /* Integrate over points within this atom's (inflated) radius */

```

```

08944     rtot = (irad + arad + thee->splineWin);
08945     rtot2 = VSQR(rtot);
08946     dx = rtot + 0.5*hx;
08947     imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
08948     imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
08949     for (i=imin; i<=imax; i++) {
08950         dx2 = VSQR(position[0] - hx*i);
08951         if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
08952         else dy = 0.5*hy;
08953         jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
08954         jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
08955         for (j=jmin; j<=jmax; j++) {
08956             dy2 = VSQR(position[1] - hy*j);
08957             if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
08958             else dz = 0.5*hzed;
08959             kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
08960             kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
08961             for (k=kmin; k<=kmax; k++) {
08962                 dz2 = VSQR(k*hzed - position[2]);
08963                 /* See if grid point is inside ivdw radius and set ccf
08964                  * accordingly (do spline assignment here) */
08965                 if ((dz2 + dy2 + dx2) <= rtot2) {
08966                     gpos[0] = i*hx + xmin;
08967                     gpos[1] = j*hy + ymin;
08968                     gpos[2] = k*hzed + zmin;
08969                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad,
08970                           atom, tgrad);
08971                     fmag = induced->data[IJK(i,j,k)];
08972                     fmag *= perm->data[IJK(i,j,k)];
08973                     fmag *= thee->kappa[IJK(i,j,k)];
08974                     force[0] += (zkappa2*fmag*tgrad[0]);
08975                     force[1] += (zkappa2*fmag*tgrad[1]);
08976                     force[2] += (zkappa2*fmag*tgrad[2]);
08977                 }
08978             } /* k loop */
08979         } /* j loop */
08980     } /* i loop */
08981 }
08982
08983     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
08984     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
08985     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
08986
08987 }
08988
08989 VPUBLIC void Vpmg_ibNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
08990                                         int atomID, double force[3]) {
08991     Vpmg_ibDirectPolForce(thee, perm, nlInduced, atomID, force);
08992 }
08993
08994 VPUBLIC void Vpmg_dbDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *induced,
08995                                         int atomID, double force[3]) {
08996
08997     Vatom *atom;
08998     Vacc *acc;
08999     Vpbe *pbe;

```

```

09000     Vsurf_Meth srfm;
09001
09002     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
09003     double xlen, ylen, zlen, xmin, ymin, xmax, ymax, zmax, rtot2, epsp;
09004     double rtot, dx, gpos[3], tgrad[3], dbFMag, epsw, kT;
09005     double *u, *up, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkml;
09006     double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
09007     double dHzijkml[3];
09008     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09009
09010     VASSERT(thee != VNULL);
09011     VASSERT(perm != VNULL); /* permanent multipole PMG solution. */
09012     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09013
09014     acc = thee->pbe->acc;
09015     atom = Valist_getAtom(thee->pbe->alist, atomID);
09016     VASSERT (atom->partID != 0); /* Currently all atoms must be in the same par-
        tition. */
09017     apos = Vatom_getPosition(atom);
09018     arad = Vatom_getRadius(atom);
09019
09020     /* Reset force */
09021     force[0] = 0.0;
09022     force[1] = 0.0;
09023     force[2] = 0.0;
09024
09025     /* Get PBE info */
09026     pbe = thee->pbe;
09027     acc = pbe->acc;
09028     srfm = thee->surfMeth;
09029     epsp = Vpbe_getSoluteDiel(pbe);
09030     epsw = Vpbe_getSolventDiel(pbe);
09031     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
09032     izmagic = 1.0/Vpbe_getZmagic(pbe);
09033
09034     deps = (epsw - epsp);
09035     depsi = 1.0/deps;
09036     VASSERT(VABS(deps) > VPMGSMALL);
09037
09038     /* Mesh info */
09039     nx = thee->pmgp->nx;
09040     ny = thee->pmgp->ny;
09041     nz = thee->pmgp->nz;
09042     hx = thee->pmgp->hx;
09043     hy = thee->pmgp->hy;
09044     hzed = thee->pmgp->hzed;
09045     xlen = thee->pmgp->xlen;
09046     ylen = thee->pmgp->ylen;
09047     zlen = thee->pmgp->zlen;
09048     xmin = thee->pmgp->xmin;
09049     ymin = thee->pmgp->ymin;
09050     zmin = thee->pmgp->zmin;
09051     xmax = thee->pmgp->xmax;
09052     ymax = thee->pmgp->ymax;
09053     zmax = thee->pmgp->zmax;
09054     /* If the permanent and induced potentials are flipped the
        results are exactly the same. */
09055

```

```

09056     u = induced->data;
09057     up = perm->data;
09058
09059     /* Make sure we're on the grid */
09060     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09061         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09062         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09063         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom at (%4.3f, %4.3f, %4.3f) is o
ff the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
09064         Vnm_print(2, "Vpmg_dbDirectPolForce: xmin = %g, xmax = %g\n", xmin, x
max);
09065         Vnm_print(2, "Vpmg_dbDirectPolForce: ymin = %g, ymax = %g\n", ymin, y
max);
09066         Vnm_print(2, "Vpmg_dbDirectPolForce: zmin = %g, zmax = %g\n", zmin, z
max);
09067         fflush(stderr);
09068     } else {
09069
09070         /* Convert the atom position to grid reference frame */
09071         position[0] = apos[0] - xmin;
09072         position[1] = apos[1] - ymin;
09073         position[2] = apos[2] - zmin;
09074
09075         /* Integrate over points within this atom's (inflated) radius */
09076         rtot = (arad + thee->splineWin);
09077         rtot2 = VSQR(rtot);
09078         dx = rtot/hx;
09079         imin = (int)floor((position[0]-rtot)/hx);
09080         if (imin < 1) {
09081             Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);

09082             return;
09083         }
09084         imax = (int)ceil((position[0]+rtot)/hx);
09085         if (imax > (nx-2)) {
09086             Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);

09087             return;
09088         }
09089         jmin = (int)floor((position[1]-rtot)/hy);
09090         if (jmin < 1) {
09091             Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);

09092             return;
09093         }
09094         jmax = (int)ceil((position[1]+rtot)/hy);
09095         if (jmax > (ny-2)) {
09096             Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);

09097             return;
09098         }
09099         kmin = (int)floor((position[2]-rtot)/hzed);
09100         if (kmin < 1) {
09101             Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);

09102             return;
09103         }

```

```

09104     kmax = (int)ceil((position[2]+rtot)/hzed);
09105     if (kmax > (nz-2)) {
09106         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);
09107         return;
09108     }
09109     for (i=iimin; i<=imax; i++) {
09110         for (j=jmin; j<=jmax; j++) {
09111             for (k=kmin; k<=kmax; k++) {
09112                 /* i,j,k */
09113                 gpos[0] = (i+0.5)*hx + xmin;
09114                 gpos[1] = j*hy + ymin;
09115                 gpos[2] = k*hzed + zmin;
09116                 Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09117                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09118                                     atom, dHxijk);
09119                 for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09120                 gpos[0] = i*hx + xmin;
09121                 gpos[1] = (j+0.5)*hy + ymin;
09122                 gpos[2] = k*hzed + zmin;
09123                 Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09124                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09125                                     atom, dHyijk);
09126                 for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09127                 gpos[0] = i*hx + xmin;
09128                 gpos[1] = j*hy + ymin;
09129                 gpos[2] = (k+0.5)*hzed + zmin;
09130                 Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09131                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09132                                     atom, dHzijk);
09133                 for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
09134                 /* i-1,j,k */
09135                 gpos[0] = (i-0.5)*hx + xmin;
09136                 gpos[1] = j*hy + ymin;
09137                 gpos[2] = k*hzed + zmin;
09138                 Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09139                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09140                                     atom, dHxim1jk);
09141                 for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
09142                 /* i,j-1,k */
09143                 gpos[0] = i*hx + xmin;
09144                 gpos[1] = (j-0.5)*hy + ymin;
09145                 gpos[2] = k*hzed + zmin;
09146                 Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09147                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09148                                     atom, dHyijm1k);
09149                 for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
09150                 /* i,j,k-1 */
09151                 gpos[0] = i*hx + xmin;
09152                 gpos[1] = j*hy + ymin;
09153                 gpos[2] = (k-0.5)*hzed + zmin;
09154                 Hzijkml = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09155                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09156                                     atom, dHzijkml);
09157                 for (l=0; l<3; l++) dHzijkml[l] *= Hzijkml;
09158
09159     dbFmag = up[IJK(i,j,k)];

```

```

09160     tgrad[0] =
09161         (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09162         + dHxim1jk[0]* (u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09163         + (dHyijk[0] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
09164         + dHyijm1k[0]* (u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09165         + (dHzijk[0] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
09166         + dHzijk1[0]* (u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09167     tgrad[1] =
09168         (dHxijk[1] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09169         + dHxim1jk[1]* (u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09170         + (dHyijk[1] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
09171         + dHyijm1k[1]* (u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09172         + (dHzijk[1] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
09173         + dHzijk1[1]* (u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09174     tgrad[2] =
09175         (dHxijk[2] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09176         + dHxim1jk[2]* (u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09177         + (dHyijk[2] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
09178         + dHyijm1k[2]* (u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09179         + (dHzijk[2] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
09180         + dHzijk1[2]* (u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09181     force[0] += (dbFmag*tgrad[0]);
09182     force[1] += (dbFmag*tgrad[1]);
09183     force[2] += (dbFmag*tgrad[2]);
09184
09185     } /* k loop */
09186     } /* j loop */
09187 } /* i loop */
09188
09189     force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
09190     force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
09191     force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
09192
09193 }
09194 }
09195
09196 VPUBLIC void Vpmg_dbNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
09197                                         int atomID, double force[3]) {
09198     Vpmg_dbDirectPolForce(thee, perm, nlInduced, atomID, force);
09199 }
09200
09201 VPUBLIC void Vpmg_qfMutualPolForce(Vpmg *thee, Vgrid *induced,
09202                                         Vgrid *nlinduced, int atomID, double force[3]) {
09203
09204     Vatom *atom;
09205     double *apos, *dipole, position[3], hx, hy, hzed;
09206     double *u, *unl;
09207     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
09208     double de[3][3], denl[3][3];
09209     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, mi, mj, mk;
09210     double ifloat, jfloat, kfloat;
09211     double f, fnl, uix, uiy, uiz, uixnl, uiynl, uiznl;
09212     int i, j, k, nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
09213     int kpl, kp2, ii, jj, kk;
09214
09215     VASSERT(thee != VNULL); /* PMG object with PBE info. */
09216     VASSERT(induced != VNULL); /* potential due to induced dipoles. */

```

```

09217     VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles. */
09218     /
09219     atom = Valist_getAtom(thee->pbe->alist, atomID);
09220     VASSERT(atom->partID != 0); /* all atoms must be in the same partition. */
09221
09222     apos = VatomGetPosition(atom);
09223     dipole = Vatom_getInducedDipole(atom);
09224     uix = dipole[0];
09225     uiy = dipole[1];
09226     uiz = dipole[2];
09227     dipole = Vatom_getNLInducedDipole(atom);
09228     uixnl = dipole[0];
09229     uiynl = dipole[1];
09230     uiznl = dipole[2];
09231     u = induced->data;
09232     unl = nlinduced->data;
09233
09234     for (i=0;i<3;i++) {
09235         for (j=0;j<3;j++) {
09236             de[i][j] = 0.0;
09237             denl[i][j] = 0.0;
09238         }
09239     /* Mesh info */
09240     nx = induced->nx;
09241     ny = induced->ny;
09242     nz = induced->nz;
09243     hx = induced->hx;
09244     hy = induced->hy;
09245     hzed = induced->hzed;
09246     xmin = induced->xmin;
09247     ymin = induced->ymin;
09248     zmin = induced->zmin;
09249     xmax = induced->xmax;
09250     ymax = induced->ymax;
09251     zmax = induced->zmax;
09252     xlen = xmax-xmin;
09253     ylen = ymax-ymin;
09254     zlen = zmax-zmin;
09255
09256     /* If we aren't in the current position, then we're done */
09257     if (atom->partID == 0) return;
09258
09259     /* Make sure we're on the grid */
09260     if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
09261         || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
09262         || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
09263         Vnm_print(2, "qfMutualPolForce: Atom off the mesh (ignoring) %6.3f %6.3f
09264             %6.3f\n", apos[0], apos[1], apos[2]);
09265         fflush(stderr);
09266     } else {
09267
09268         /* Convert the atom position to grid coordinates */
09269         position[0] = apos[0] - xmin;
09270         position[1] = apos[1] - ymin;
09271         position[2] = apos[2] - zmin;

```

```

09271     ifloat = position[0]/hx;
09272     jfloat = position[1]/hy;
09273     kfloat = position[2]/hzed;
09274     ip1 = (int)ceil(ifloat);
09275     ip2 = ip1 + 2;
09276     im1 = (int)floor(ifloat);
09277     im2 = im1 - 2;
09278     jp1 = (int)ceil(jfloat);
09279     jp2 = jp1 + 2;
09280     jm1 = (int)floor(jfloat);
09281     jm2 = jm1 - 2;
09282     kp1 = (int)ceil(kfloat);
09283     kp2 = kp1 + 2;
09284     km1 = (int)floor(kfloat);
09285     km2 = km1 - 2;
09286
09287     /* This step shouldn't be necessary, but it saves nasty debugging
09288      * later on if something goes wrong */
09289     ip2 = VMIN2(ip2,nx-1);
09290     ip1 = VMIN2(ip1,nx-1);
09291     im1 = VMAX2(im1,0);
09292     im2 = VMAX2(im2,0);
09293     jp2 = VMIN2(jp2,ny-1);
09294     jp1 = VMIN2(jp1,ny-1);
09295     jm1 = VMAX2(jm1,0);
09296     jm2 = VMAX2(jm2,0);
09297     kp2 = VMIN2(kp2,nz-1);
09298     kp1 = VMIN2(kp1,nz-1);
09299     km1 = VMAX2(km1,0);
09300     km2 = VMAX2(km2,0);
09301
09302     for (ii=im2; ii<ip2; ii++) {
09303         mi = VFCHI4(ii,ifloat);
09304         mx = bspline4(mi);
09305         dmx = dbspline4(mi);
09306         d2mx = d2bspline4(mi);
09307         for (jj=jm2; jj<=jp2; jj++) {
09308             mj = VFCHI4(jj,jfloat);
09309             my = bspline4(mj);
09310             dmy = dbspline4(mj);
09311             d2my = d2bspline4(mj);
09312             for (kk=km2; kk<=kp2; kk++) {
09313                 mk = VFCHI4(kk,kfloat);
09314                 mz = bspline4(mk);
09315                 dmz = dbspline4(mk);
09316                 d2mz = d2bspline4(mk);
09317                 f = u[IJK(ii,jj,kk)];
09318                 fnl = unl[IJK(ii,jj,kk)];
09319
09320                 /* The gradient of the reaction field
09321                  due to induced dipoles */
09322                 de[0][0] += f*d2mx*my*mz/(hx*hx);
09323                 de[1][0] += f*dmx*dmy*mz/(hy*hy);
09324                 de[1][1] += f*mx*d2my*mz/(hy*hy);
09325                 de[2][0] += f*dmx*my*dmz/(hx*hzed);
09326                 de[2][1] += f*mx*dmy*dmz/(hy*hzed);
09327                 de[2][2] += f*mx*my*d2mz/(hzed*hzed);

```

```

09328
09329      /* The gradient of the reaction field
09330      due to non-local induced dipoles */
09331      denl[0][0] += fnl*d2mx*my*mz/(hx*hx);
09332      denl[1][0] += fnl*dmx*dmy*mz/(hy*hy);
09333      denl[1][1] += fnl*mx*d2my*mz/(hy*hy);
09334      denl[2][0] += fnl*dmx*my*dmz/(hx*hzed);
09335      denl[2][1] += fnl*mx*dmy*dmz/(hy*hzed);
09336      denl[2][2] += fnl*mx*my*d2mz/(hzed*hzed);
09337      }
09338      }
09339      }
09340      }
09341
09342      /* mutual polarization force */
09343      force[0] = -(de[0][0]*uixnl + de[1][0]*uiynl + de[2][0]*uiznl);
09344      force[1] = -(de[1][0]*uixnl + de[1][1]*uiynl + de[2][1]*uiznl);
09345      force[2] = -(de[2][0]*uixnl + de[2][1]*uiynl + de[2][2]*uiznl);
09346      force[0] -= denl[0][0]*uix + denl[1][0]*uiy + denl[2][0]*uiz;
09347      force[1] -= denl[1][0]*uix + denl[1][1]*uiy + denl[2][1]*uiz;
09348      force[2] -= denl[2][0]*uix + denl[2][1]*uiy + denl[2][2]*uiz;
09349
09350      force[0] = 0.5 * force[0];
09351      force[1] = 0.5 * force[1];
09352      force[2] = 0.5 * force[2];
09353
09354  }
09355
09356 VPUBLIC void Vpmg_ibMutualPolForce(Vpmg *thee, Vgrid *induced, Vgrid *nlinduced,
09357                           int atomID, double force[3]) {
09358
09359     Vatom *atom;
09360     Valist *alist;
09361     Vacc *acc;
09362     Vpbe *pbe;
09363     Vsurf_Meth srfm;
09364
09365     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
09366     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
09367     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
09368     double izmagic;
09369     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09370
09371     VASSERT(thee != VNULL);           /* We need a PMG object with PBE info. */
09372     VASSERT(induced != VNULL);        /* We need the potential due to induced dipole
09373     s. */
09374     VASSERT(nlinduced != VNULL);       /* We need the potential due to non-local induced dipoles. */
09375     VASSERT (!thee->pmpg->nonlin); /* Nonlinear PBE is not implemented for AMOEBA
09376     */
09377
09378     atom = Valist_getAtom(thee->pbe->alist, atomID);
09379     VASSERT (atom->partID != 0);    /* Currently all atoms must be in the same partition. */
09380     acc = thee->pbe->acc;
09381     srfm = thee->surfMeth;

```

```

09381     apos = Vatom_getPosition(atom);
09382     arad = Vatom_getRadius(atom);
09383
09384     /* Reset force */
09385     force[0] = 0.0;
09386     force[1] = 0.0;
09387     force[2] = 0.0;
09388
09389     /* If we aren't in the current position, then we're done */
09390     if (atom->partID == 0) return;
09391
09392     /* Get PBE info */
09393     pbe = theee->pbe;
09394     acc = pbe->acc;
09395     alist = pbe->alist;
09396     irad = Vpbe_getMaxIonRadius(pbe);
09397     zkappa2 = Vpbe_getZkappa2(pbe);
09398     izmagic = 1.0/Vpbe_getZmagic(pbe);
09399
09400     VASSERT (zkappa2 > VPMGSMALL); /* Should be a check for this further up.*/
09401
09402     /* Mesh info */
09403     nx = induced->nx;
09404     ny = induced->ny;
09405     nz = induced->nz;
09406     hx = induced->hx;
09407     hy = induced->hy;
09408     hzed = induced->hzed;
09409     xmin = induced->xmin;
09410     ymin = induced->ymin;
09411     zmin = induced->zmin;
09412     xmax = induced->xmax;
09413     ymax = induced->ymax;
09414     zmax = induced->zmax;
09415     xlen = xmax-xmin;
09416     ylen = ymax-ymin;
09417     zlen = zmax-zmin;
09418
09419     /* Make sure we're on the grid */
09420     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09421         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09422         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09423         Vnm_print(2, "Vpmg_ibMutalPolForce: Atom at (%4.3f, %4.3f, %4.3f) is off
the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
09424         Vnm_print(2, "Vpmg_ibMutalPolForce: xmin = %g, xmax = %g\n", xmin, xmax);
09425         Vnm_print(2, "Vpmg_ibMutalPolForce: ymin = %g, ymax = %g\n", ymin, ymax);
09426         Vnm_print(2, "Vpmg_ibMutalPolForce: zmin = %g, zmax = %g\n", zmin, zmax);
09427         fflush(stderr);
09428     } else {
09429
09430         /* Convert the atom position to grid reference frame */
09431         position[0] = apos[0] - xmin;
09432         position[1] = apos[1] - ymin;
09433         position[2] = apos[2] - zmin;

```

```

09434
09435     /* Integrate over points within this atom's (inflated) radius */
09436     rtot = (irad + arad + thee->splineWin);
09437     rtot2 = VSQR(rtot);
09438     dx = rtot + 0.5*hx;
09439     imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
09440     imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
09441     for (i=imin; i<=imax; i++) {
09442         dx2 = VSQR(position[0] - hx*i);
09443         if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
09444         else dy = 0.5*hy;
09445         jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
09446         jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
09447         for (j=jmin; j<=jmax; j++) {
09448             dy2 = VSQR(position[1] - hy*j);
09449             if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
09450             else dz = 0.5*hzed;
09451             kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
09452             kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
09453             for (k=kmin; k<=kmax; k++) {
09454                 dz2 = VSQR(k*hzed - position[2]);
09455                 /* See if grid point is inside ivdw radius and set ccf
09456                  * accordingly (do spline assignment here) */
09457                 if ((dz2 + dy2 + dx2) <= rtot2) {
09458                     gpos[0] = i*hx + xmin;
09459                     gpos[1] = j*hy + ymin;
09460                     gpos[2] = k*hzed + zmin;
09461                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad,
09462
09463                         atom, tgrad);
09464                     fmag = induced->data[IJK(i,j,k)];
09465                     fmag *= nlinduced->data[IJK(i,j,k)];
09466                     fmag *= thee->kappa[IJK(i,j,k)];
09467                     force[0] += (zkappa2*fmag*tgrad[0]);
09468                     force[1] += (zkappa2*fmag*tgrad[1]);
09469                     force[2] += (zkappa2*fmag*tgrad[2]);
09470                 }
09471             } /* k loop */
09472         } /* j loop */
09473     } /* i loop */
09474
09475     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
09476     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
09477     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
09478 }
09479
09480 VPUBLIC void Vpmg_dbMutualPolForce(Vpmg *thee, Vgrid *induced,
09481                                         Vgrid *nlinduced, int atomID,
09482                                         double force[3]) {
09483
09484     Vatom *atom;
09485     Vacc *acc;
09486     Vpbe *pbe;
09487     Vsurf_Meth srfm;
09488
09489     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;

```

```

09490     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
09491     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
09492     double *u, *unl, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkml;
09493     double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
09494     double dHzijkml[3];
09495     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09496
09497     VASSERT(thee != VNULL); /* PMG object with PBE info. */
09498     VASSERT(induced != VNULL); /* potential due to induced dipoles.*/
09499     VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles.*/
09500
09501     acc = thee->pbe->acc;
09502     srfm = thee->surfMeth;
09503     atom = Valist_getAtom(thee->pbe->alist, atomID);
09504     VASSERT (atom->partID != 0); /* all atoms must be in the same partition.*/
09505     apos = Vatom_getPosition(atom);
09506     arad = Vatom_getRadius(atom);
09507
09508     /* Reset force */
09509     force[0] = 0.0;
09510     force[1] = 0.0;
09511     force[2] = 0.0;
09512
09513     /* Get PBE info */
09514     pbe = thee->pbe;
09515     acc = pbe->acc;
09516     epsp = Vpbe_getSoluteDiel(pbe);
09517     epsw = Vpbe_getSolventDiel(pbe);
09518     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
09519     izmagic = 1.0/Vpbe_getZmagic(pbe);
09520
09521     deps = (epsw - epsp);
09522     depsi = 1.0/deps;
09523     VASSERT(VABS(deps) > VPMGSMALL);
09524
09525     /* Mesh info */
09526     nx = thee->pmgp->nx;
09527     ny = thee->pmgp->ny;
09528     nz = thee->pmgp->nz;
09529     hx = thee->pmgp->hx;
09530     hy = thee->pmgp->hy;
09531     hzed = thee->pmgp->hzed;
09532     xlen = thee->pmgp->xlen;
09533     ylen = thee->pmgp->ylen;
09534     zlen = thee->pmgp->zlen;
09535     xmin = thee->pmgp->xmin;
09536     ymin = thee->pmgp->ymin;
09537     zmin = thee->pmgp->zmin;
09538     xmax = thee->pmgp->xmax;
09539     ymax = thee->pmgp->ymax;
09540     zmax = thee->pmgp->zmax;
09541     u = induced->data;
09542     unl = nlinduced->data;
09543
09544     /* Make sure we're on the grid */
09545     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \

```

```

09546     (apos[1]<=ymin) || (apos[1]>=ymax) || \
09547     (apos[2]<=zmin) || (apos[2]>=zmax)) {
09548         Vnm_print(2, "Vpmg_dbMutualPolForce: Atom at (%4.3f, %4.3f, %4.3f) is off \
09549 the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
09550         Vnm_print(2, "Vpmg_dbMutualPolForce: xmin = %g, xmax = %g\n", xmin, xmax);
09551         Vnm_print(2, "Vpmg_dbMutualPolForce: ymin = %g, ymax = %g\n", ymin, ymax);
09552         Vnm_print(2, "Vpmg_dbMutualPolForce: zmin = %g, zmax = %g\n", zmin, zmax);
09553         fflush(stderr);
09554     } else {
09555         /* Convert the atom position to grid reference frame */
09556         position[0] = apos[0] - xmin;
09557         position[1] = apos[1] - ymin;
09558         position[2] = apos[2] - zmin;
09559
09560         /* Integrate over points within this atom's (inflated) radius */
09561         rtot = (arad + thee->splineWin);
09562         rtot2 = VSQR(rtot);
09563         dx = rtot/hx;
09564         imin = (int)floor((position[0]-rtot)/hx);
09565         if (imin < 1) {
09566             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09567
09568             return;
09569         }
09570         imax = (int)ceil((position[0]+rtot)/hx);
09571         if (imax > (nx-2)) {
09572             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09573
09574             return;
09575         }
09576         jmin = (int)floor((position[1]-rtot)/hy);
09577         if (jmin < 1) {
09578             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09579
09580             return;
09581         }
09582         jmax = (int)ceil((position[1]+rtot)/hy);
09583         if (jmax > (ny-2)) {
09584             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09585
09586             return;
09587         }
09588         kmin = (int)floor((position[2]-rtot)/hzed);
09589         if (kmin < 1) {
09590             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09591
09592             return;

```

```

09593
09594     }
09595     for (i=iimin; i<=imax; i++) {
09596         for (j=jmin; j<=jmax; j++) {
09597             for (k=kmin; k<=kmax; k++) {
09598                 /* i, j, k */
09599                 gpos[0] = (i+0.5)*hx + xmin;
09600                 gpos[1] = j*hy + ymin;
09601                 gpos[2] = k*hzed + zmin;
09602                 Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09603                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09604                     atom, dHxijk);
09605                 for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09606                 gpos[0] = i*hx + xmin;
09607                 gpos[1] = (j+0.5)*hy + ymin;
09608                 gpos[2] = k*hzed + zmin;
09609                 Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09610                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09611                     atom, dHyijk);
09612                 for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09613                 gpos[0] = i*hx + xmin;
09614                 gpos[1] = j*hy + ymin;
09615                 gpos[2] = (k+0.5)*hzed + zmin;
09616                 Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09617                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09618                     atom, dHzijk);
09619                 /* i-1, j, k */
09620                 gpos[0] = (i-0.5)*hx + xmin;
09621                 gpos[1] = j*hy + ymin;
09622                 gpos[2] = k*hzed + zmin;
09623                 Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09624                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09625                     atom, dHxim1jk);
09626                 for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
09627                 /* i, j-1, k */
09628                 gpos[0] = i*hx + xmin;
09629                 gpos[1] = (j-0.5)*hy + ymin;
09630                 gpos[2] = k*hzed + zmin;
09631                 Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09632                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09633                     atom, dHyijm1k);
09634                 for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
09635                 /* i, j, k-1 */
09636                 gpos[0] = i*hx + xmin;
09637                 gpos[1] = j*hy + ymin;
09638                 gpos[2] = (k-0.5)*hzed + zmin;
09639                 Hzijkml = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09640                 Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09641                     atom, dHzijkml);
09642                 for (l=0; l<3; l++) dHzijkml[l] *= Hzijkml;
09643                 dbFmag = unl[IJK(i,j,k)];
09644                 tgrad[0] =
09645                     (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09646                     + dHxim1jk[0]* (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
09647                     + (dHyijk[0] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09648                     + dHyijm1k[0]* (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
09649                     + (dHzijk[0] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))

```

```

09650     + dHzijkml[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09651 tgrad[1] =
09652     (dHxijk[1]* (u[IJK(i+1,j,k)]-u[IJK(i,j,k)]))
09653     + dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09654     + (dHyijk[1]* (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
09655     + dHyjm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09656     + (dHzijk[1]* (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
09657     + dHzijkml[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09658 tgrad[2] =
09659     (dHxijk[2]* (u[IJK(i+1,j,k)]-u[IJK(i,j,k)]))
09660     + dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09661     + (dHyijk[2]* (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
09662     + dHyjm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09663     + (dHzijk[2]* (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
09664     + dHzijkml[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09665 force[0] += (dbFmag*tgrad[0]);
09666 force[1] += (dbFmag*tgrad[1]);
09667 force[2] += (dbFmag*tgrad[2]);
09668 } /* k loop */
09669 } /* j loop */
09670 } /* i loop */
09671
09672 force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
09673 force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
09674 force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
09675 }
09676 }
09677
09678 #endif /* if defined(WITH_TINKER) */
09679
09680 VPRIPRIVATE void fillcoCoefSpline4(Vpmg *thee) {
09681     Valist *alist;
09682     Vpbe *pbe;
09683     Vatom *atom;
09684     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2,
09685     double xlabel, ylabel, zlabel, position[3], itot, stot, ictot, ictot2, sctot;
09686     double irad, dx, dy, dz, epsw, epsp, w2i;
09687     double hx, hy, hzed, *apos, arad, sctot2;
09688     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
09689     double dist, value, denom, sm, sm2, sm3, sm4, sm5, sm6, sm7;
09690     double e, e2, e3, e4, e5, e6, e7;
09691     double b, b2, b3, b4, b5, b6, b7;
09692     double c0, c1, c2, c3, c4, c5, c6, c7;
09693     double ic0, ic1, ic2, ic3, ic4, ic5, ic6, ic7;
09694     int i, j, k, nx, ny, nz, atom;
09695     int imin, imax, jmin, jmax, kmin, kmax;
09696
09697     VASSERT(thee != VNULL);
09698     splineWin = thee->splineWin;
09699
09700     /* Get PBE info */
09701     pbe = thee->pbe;
09702     alist = pbe->alist;
09703     irad = Vpbe_getMaxIonRadius(pbe);
09704     ionstr = Vpbe_getBulkIonicStrength(pbe);
09705     epsw = Vpbe_getSolventDiel(pbe);

```

```

09707     epss = Vpbe_getSoluteDiel(pbe);
09708
09709     /* Mesh info */
09710     nx = thee->pmgp->nx;
09711     ny = thee->pmgp->ny;
09712     nz = thee->pmgp->nz;
09713     hx = thee->pmgp->hx;
09714     hy = thee->pmgp->hy;
09715     hzed = thee->pmgp->hzed;
09716
09717     /* Define the total domain size */
09718     xlabel = thee->pmgp->xlen;
09719     ylabel = thee->pmgp->ylen;
09720     zlabel = thee->pmgp->zlen;
09721
09722     /* Define the min/max dimensions */
09723     xmin = thee->pmgp->xcent - (xlen/2.0);
09724     ymin = thee->pmgp->ycent - (ylen/2.0);
09725     zmin = thee->pmgp->zcent - (zlen/2.0);
09726     xmax = thee->pmgp->xcent + (xlen/2.0);
09727     ymax = thee->pmgp->ycent + (ylen/2.0);
09728     zmax = thee->pmgp->zcent + (zlen/2.0);
09729
09730     /* This is a floating point parameter related to the non-zero nature of the
09731      * bulk ionic strength. If the ionic strength is greater than zero; this
09732      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
09733      * Otherwise, this parameter is set to 0.0 */
09734     if (ionstr > VPMGSMALL) ionmask = 1.0;
09735     else ionmask = 0.0;
09736
09737     /* Reset the kappa, epsx, epsy, and epsz arrays */
09738     for (i=0; i<(nx*ny*nz); i++) {
09739         thee->kappa[i] = 1.0;
09740         thee->epsx[i] = 1.0;
09741         thee->epsy[i] = 1.0;
09742         thee->epsz[i] = 1.0;
09743     }
09744
09745     /* Loop through the atoms and do assign the dielectric */
09746     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
09747
09748         atom = Valist_getAtom(alist, iatom);
09749         apos = Vatom_getPosition(atom);
09750         arad = Vatom_getRadius(atom);
09751
09752         b = arad - splineWin;
09753         e = arad + splineWin;
09754         e2 = e * e;
09755         e3 = e2 * e;
09756         e4 = e3 * e;
09757         e5 = e4 * e;
09758         e6 = e5 * e;
09759         e7 = e6 * e;
09760         b2 = b * b;
09761         b3 = b2 * b;
09762         b4 = b3 * b;
09763         b5 = b4 * b;

```

```

09764     b6 = b5 * b;
09765     b7 = b6 * b;
09766     denom = e7 - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
09767         + 35.0*e3*b4 - 21.0*b5*e2 + 7.0*e*b6 - b7;
09768     c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
09769     c1 = -140.0*b3*e3/denom;
09770     c2 = 210.0*e2*b2*(e + b)/denom;
09771     c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
09772     c4 = 35.0*(e3 + 9.0*b*e2 + 9.0*e*b2 + b3)/denom;
09773     c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
09774     c6 = 70.0*(e + b)/denom;
09775     c7 = -20.0/denom;
09776
09777     b = irad + arad - splineWin;
09778     e = irad + arad + splineWin;
09779     e2 = e * e;
09780     e3 = e2 * e;
09781     e4 = e3 * e;
09782     e5 = e4 * e;
09783     e6 = e5 * e;
09784     e7 = e6 * e;
09785     b2 = b * b;
09786     b3 = b2 * b;
09787     b4 = b3 * b;
09788     b5 = b4 * b;
09789     b6 = b5 * b;
09790     b7 = b6 * b;
09791     denom = e7 - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
09792         + 35.0*e3*b4 - 21.0*b5*e2 + 7.0*e*b6 - b7;
09793     ic0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
09794     ic1 = -140.0*b3*e3/denom;
09795     ic2 = 210.0*e2*b2*(e + b)/denom;
09796     ic3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
09797     ic4 = 35.0*(e3 + 9.0*b*e2 + 9.0*e*b2 + b3)/denom;
09798     ic5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
09799     ic6 = 70.0*(e + b)/denom;
09800     ic7 = -20.0/denom;
09801
09802     /* Make sure we're on the grid */
09803     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09804         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09805         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09806         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
09807             (thee->pmgp->bcfl != BCFL_MAP)) {
09808             Vnm_print(2, "Vpmg_fillco: Atom %d at (%4.3f, %4.3f,\n
09809 %4.3f) is off the mesh (ignoring):\n",
09810                 iatom, apos[0], apos[1], apos[2]);
09811             Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
09812                 xmin, xmax);
09813             Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
09814                 ymin, ymax);
09815             Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
09816                 zmin, zmax);
09817         }
09818         fflush(stderr);
09819     } else if (arad > VPMGSMALL) { /* if we're on the mesh */

```

```

09821
09822     /* Convert the atom position to grid reference frame */
09823     position[0] = apos[0] - xmin;
09824     position[1] = apos[1] - ymin;
09825     position[2] = apos[2] - zmin;
09826
09827     /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
09828      * ASSIGNMENT (Steps #1-3) */
09829     itot = irad + arad + splineWin;
09830     itot2 = VSQR(itot);
09831     ictot = VMAX2(0, (irad + arad - splineWin));
09832     ictot2 = VSQR(ictot);
09833     stot = arad + splineWin;
09834     stot2 = VSQR(stot);
09835     sctot = VMAX2(0, (arad - splineWin));
09836     sctot2 = VSQR(sctot);
09837
09838     /* We'll search over grid points which are in the greater of
09839      * these two radii */
09840     rtot = VMAX2(itot, stot);
09841     rtot2 = VMAX2(itot2, stot2);
09842     dx = rtot + 0.5*hx;
09843     dy = rtot + 0.5*hy;
09844     dz = rtot + 0.5*hzed;
09845     imin = VMAX2(0, (int)floor((position[0] - dx)/hx));
09846     imax = VMIN2(nx-1, (int)ceil((position[0] + dx)/hx));
09847     jmin = VMAX2(0, (int)floor((position[1] - dy)/hy));
09848     jmax = VMIN2(ny-1, (int)ceil((position[1] + dy)/hy));
09849     kmin = VMAX2(0, (int)floor((position[2] - dz)/hzed));
09850     kmax = VMIN2(nz-1, (int)ceil((position[2] + dz)/hzed));
09851     for (i=imin; i<=imax; i++) {
09852         dx2 = VSQR(position[0] - hx*i);
09853         for (j=jmin; j<=jmax; j++) {
09854             dy2 = VSQR(position[1] - hy*j);
09855             for (k=kmin; k<=kmax; k++) {
09856                 dz2 = VSQR(position[2] - k*hzed);
09857
09858                 /* ASSIGN CCF */
09859                 if (thee->kappa[IJK(i, j, k)] > VPMGSMALL) {
09860                     dist2 = dz2 + dy2 + dx2;
09861                     if (dist2 >= itot2) {
09862                         ;
09863                     }
09864                     if (dist2 <= ictot2) {
09865                         thee->kappa[IJK(i, j, k)] = 0.0;
09866                     }
09867                     if ((dist2 < itot2) && (dist2 > ictot2)) {
09868                         dist = VSQRT(dist2);
09869                         sm = dist;
09870                         sm2 = dist2;
09871                         sm3 = sm2 * sm;
09872                         sm4 = sm3 * sm;
09873                         sm5 = sm4 * sm;
09874                         sm6 = sm5 * sm;
09875                         sm7 = sm6 * sm;
09876                         value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
09877                             + ic4*sm4 + ic5*sm5 + ic6*sm6 + ic7*sm7;

```

```

09878                     if (value > 1.0) {
09879                         value = 1.0;
09880                     } else if (value < 0.0){
09881                         value = 0.0;
09882                     }
09883                     theee->kappa[IJK(i,j,k)] *= value;
09884                 }
09885             }
09886
09887             /* ASSIGN A1CF */
09888             if (theee->epsx[IJK(i,j,k)] > VPMGSMALL) {
09889                 dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
09890                 if (dist2 >= stot2) {
09891                     theee->epsx[IJK(i,j,k)] *= 1.0;
09892                 }
09893                 if (dist2 <= stot2) {
09894                     theee->epsx[IJK(i,j,k)] = 0.0;
09895                 }
09896                 if ((dist2 > stot2) && (dist2 < stot2)) {
09897                     dist = VSQRT(dist2);
09898                     sm = dist;
09899                     sm2 = VSQR(sm);
09900                     sm3 = sm2 * sm;
09901                     sm4 = sm3 * sm;
09902                     sm5 = sm4 * sm;
09903                     sm6 = sm5 * sm;
09904                     sm7 = sm6 * sm;
09905                     value = c0 + c1*sm + c2*sm2 + c3*sm3
09906                         + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
09907                     if (value > 1.0) {
09908                         value = 1.0;
09909                     } else if (value < 0.0){
09910                         value = 0.0;
09911                     }
09912                     theee->epsx[IJK(i,j,k)] *= value;
09913                 }
09914             }
09915
09916             /* ASSIGN A2CF */
09917             if (theee->epsy[IJK(i,j,k)] > VPMGSMALL) {
09918                 dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
09919                 if (dist2 >= stot2) {
09920                     theee->epsy[IJK(i,j,k)] *= 1.0;
09921                 }
09922                 if (dist2 <= stot2) {
09923                     theee->epsy[IJK(i,j,k)] = 0.0;
09924                 }
09925                 if ((dist2 > stot2) && (dist2 < stot2)) {
09926                     dist = VSQRT(dist2);
09927                     sm = dist;
09928                     sm2 = VSQR(sm);
09929                     sm3 = sm2 * sm;
09930                     sm4 = sm3 * sm;
09931                     sm5 = sm4 * sm;
09932                     sm6 = sm5 * sm;
09933                     sm7 = sm6 * sm;
09934                     value = c0 + c1*sm + c2*sm2 + c3*sm3

```

```

09935           + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
09936     if (value > 1.0) {
09937       value = 1.0;
09938     } else if (value < 0.0){
09939       value = 0.0;
09940     }
09941     thee->epsy[IJK(i,j,k)] *= value;
09942   }
09943 }
09944
09945 /* ASSIGN A3CF */
09946 if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
09947   dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
09948   if (dist2 >= stot2) {
09949     thee->epsz[IJK(i,j,k)] *= 1.0;
09950   }
09951   if (dist2 <= sctot2) {
09952     thee->epsz[IJK(i,j,k)] = 0.0;
09953   }
09954   if ((dist2 > sctot2) && (dist2 < stot2)) {
09955     dist = VSQRT(dist2);
09956     sm = dist;
09957     sm2 = dist2;
09958     sm3 = sm2 * sm;
09959     sm4 = sm3 * sm;
09960     sm5 = sm4 * sm;
09961     sm6 = sm5 * sm;
09962     sm7 = sm6 * sm;
09963     value = c0 + c1*sm + c2*sm2 + c3*sm3
09964           + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
09965     if (value > 1.0) {
09966       value = 1.0;
09967     } else if (value < 0.0){
09968       value = 0.0;
09969     }
09970     thee->epsz[IJK(i,j,k)] *= value;
09971   }
09972 }
09973
09974
09975   } /* k loop */
09976 } /* j loop */
09977 } /* i loop */
09978 } /* endif (on the mesh) */
09979 } /* endfor (over all atoms) */
09980
09981 Vnm_print(0, "Vpmg_fillco: filling coefficient arrays\n");
09982 /* Interpret markings and fill the coefficient arrays */
09983 for (k=0; k<nz; k++) {
09984   for (j=0; j<ny; j++) {
09985     for (i=0; i<nx; i++) {
09986
09987       thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
09988       thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
09989         + epsp;
09990       thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
09991         + epsp;

```

```

09992         thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
09993             + epsp;
09994
09995     } /* i loop */
09996     } /* j loop */
09997 } /* k loop */
09998
09999 }
10000
10001 VPUBLIC void fillcoPermanentInduced(Vpmg *thee) {
10002
10003     Valist *alist;
10004     Vpbe *pbe;
10005     Vatom *atom;
10006     /* Coversions */
10007     double zmagic, f;
10008     /* Grid */
10009     double xmin, xmax, ymin, ymax, zmin, zmax;
10010    double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
10011    double hx, hy, hzed, *apos;
10012    /* Multipole */
10013    double charge, *dipole,*quad;
10014    double c,ux,uy,uz,qxx,qyx,qyy,qzx,qzy,qzz,qave;
10015    /* B-spline weights */
10016    double mx,my,mz,dmx,dmy,dmz,d2mx,d2my,d2mz;
10017    double mi,mj,mk;
10018    /* Loop variables */
10019    int i, ii, jj, kk, nx, ny, nz, iatom;
10020    int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
10021
10022    VASSERT(thee != VNULL);
10023
10024    /* Get PBE info */
10025    pbe = thee->pbe;
10026    alist = pbe->alist;
10027    zmagic = Vpbe_getZmagic(pbe);
10028
10029    /* Mesh info */
10030    nx = thee->pmgp->nx;
10031    ny = thee->pmgp->ny;
10032    nz = thee->pmgp->nz;
10033    hx = thee->pmgp->hx;
10034    hy = thee->pmgp->hy;
10035    hzed = thee->pmgp->hzed;
10036
10037    /* Conversion */
10038    f = zmagic/(hx*hy*hzed);
10039
10040    /* Define the total domain size */
10041    xlen = thee->pmgp->xlen;
10042    ylen = thee->pmgp->ylen;
10043    zlen = thee->pmgp->zlen;
10044
10045    /* Define the min/max dimensions */
10046    xmin = thee->pmgp->xcent - (xlen/2.0);
10047    ymin = thee->pmgp->ycent - (ylen/2.0);
10048    zmin = thee->pmgp->zcent - (zlen/2.0);

```

```

10049     xmax = thee->pmgp->xcent + (xlen/2.0);
10050     ymax = thee->pmgp->ycent + (ylen/2.0);
10051     zmax = thee->pmgp->zcent + (zlen/2.0);
10052
10053     /* Fill in the source term (permanent atomic multipoles
10054        and induced dipoles) */
10055     Vnm_print(0, "fillcoPermanentInduced: filling in source term.\n");
10056     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
10057
10058         atom = Valist_getAtom(alist, iatom);
10059         apos = Vatom_getPosition(atom);
10060
10061         c = Vatom_getCharge(atom)*f;
10062
10063 #if defined(WITH_TINKER)
10064     dipole = Vatom_getDipole(atom);
10065     ux = dipole[0]/hx*f;
10066     uy = dipole[1]/hy*f;
10067     uz = dipole[2]/hzed*f;
10068     dipole = Vatom_getInducedDipole(atom);
10069     ux = ux + dipole[0]/hx*f;
10070     uy = uy + dipole[1]/hy*f;
10071     uz = uz + dipole[2]/hzed*f;
10072     quad = Vatom_getQuadrupole(atom);
10073     qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
10074     qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
10075     qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
10076     qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
10077     qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
10078     qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
10079 #else
10080     ux = 0.0;
10081     uy = 0.0;
10082     uz = 0.0;
10083     qxx = 0.0;
10084     qyx = 0.0;
10085     qyy = 0.0;
10086     qzx = 0.0;
10087     qzy = 0.0;
10088     qzz = 0.0;
10089 #endif /* if defined(WITH_TINKER) */
10090
10091     /* Make sure we're on the grid */
10092     if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
10093         (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
10094         (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
10095         Vnm_print(2, "fillcoPermanentMultipole: Atom #%d at (%4.3f, %4.3f, %4
.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
10096
10097         Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin
, xmax);
10098         Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin
, ymax);
10099         Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin
, zmax);
10100         fflush(stderr);
} else {

```

```

10101
10102             /* Convert the atom position to grid reference frame */
10103     position[0] = apos[0] - xmin;
10104     position[1] = apos[1] - ymin;
10105     position[2] = apos[2] - zmin;
10106
10107             /* Figure out which vertices we're next to */
10108     ifloat = position[0]/hx;
10109     jfloat = position[1]/hy;
10110     kfloat = position[2]/hzed;
10111
10112     ip1    = (int)ceil(ifloat);
10113     ip2    = ip1 + 2;
10114     im1    = (int)floor(ifloat);
10115     im2    = im1 - 2;
10116     jp1    = (int)ceil(jfloat);
10117     jp2    = jp1 + 2;
10118     jm1    = (int)floor(jfloat);
10119     jm2    = jm1 - 2;
10120     kp1    = (int)ceil(kfloat);
10121     kp2    = kp1 + 2;
10122     km1    = (int)floor(kfloat);
10123     km2    = km1 - 2;
10124
10125             /* This step shouldn't be necessary, but it saves nasty debugging
10126             * later on if something goes wrong */
10127     ip2 = VMIN2(ip2,nx-1);
10128     ip1 = VMIN2(ip1,nx-1);
10129     im1 = VMAX2(im1,0);
10130     im2 = VMAX2(im2,0);
10131     jp2 = VMIN2(jp2,ny-1);
10132     jp1 = VMIN2(jp1,ny-1);
10133     jm1 = VMAX2(jm1,0);
10134     jm2 = VMAX2(jm2,0);
10135     kp2 = VMIN2(kp2,nz-1);
10136     kp1 = VMIN2(kp1,nz-1);
10137     km1 = VMAX2(km1,0);
10138     km2 = VMAX2(km2,0);
10139
10140             /* Now assign fractions of the charge to the nearby verts */
10141     for (ii=im2; ii<=ip2; ii++) {
10142         mi = VFCHI4(ii,ifloat);
10143         mx = bspline4(mi);
10144         dmx = dbspline4(mi);
10145         d2mx = d2bspline4(mi);
10146         for (jj=jm2; jj<=jp2; jj++) {
10147             mj = VFCHI4(jj,jfloat);
10148             my = bspline4(mj);
10149             dmy = dbspline4(mj);
10150             d2my = d2bspline4(mj);
10151             for (kk=km2; kk<=kp2; kk++) {
10152                 mk = VFCHI4(kk,kfloat);
10153                 mz = bspline4(mk);
10154                 dmz = dbspline4(mk);
10155                 d2mz = d2bspline4(mk);
10156                 charge = mx*my*mz*c -
10157                         dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +

```

```

10158             d2mx*my*mz*qxx +
10159             dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
10160             dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
10161             thee->charge[IJK(ii,jj,kk)] += charge;
10162         }
10163     }
10164   }
10165 }
10166 } /* endif (on the mesh) */
10167 } /* endfor (each atom) */
10168 }
10169 }
10170
10171 VPRIVATE void fillcoCoefSpline3(Vpmg *thee) {
10172
10173   Valist *alist;
10174   Vpbe *pbe;
10175   Vatom *atom;
10176   double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
10177   double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
10178   double irad, dx, dy, dz, epsw, epss, w2i;
10179   double hx, hy, hzed, *apos, arad, sctot2;
10180   double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
10181   double dist, value, denom, sm, sm2, sm3, sm4, sm5;
10182   double e, e2, e3, e4, e5;
10183   double b, b2, b3, b4, b5;
10184   double c0, c1, c2, c3, c4, c5;
10185   double ic0, ic1, ic2, ic3, ic4, ic5;
10186   int i, j, k, nx, ny, nz, iatom;
10187   int imin, imax, jmin, jmax, kmin, kmax;
10188
10189   VASSERT(thee != VNULL);
10190   splineWin = thee->splineWin;
10191
10192   /* Get PBE info */
10193   pbe = thee->pbe;
10194   alist = pbe->alist;
10195   irad = Vpbe_getMaxIonRadius(pbe);
10196   ionstr = Vpbe_getBulkIonicStrength(pbe);
10197   epsw = Vpbe_getSolventDiel(pbe);
10198   epss = Vpbe_getSoluteDiel(pbe);
10199
10200   /* Mesh info */
10201   nx = thee->pmgp->nx;
10202   ny = thee->pmgp->ny;
10203   nz = thee->pmgp->nz;
10204   hx = thee->pmgp->hx;
10205   hy = thee->pmgp->hy;
10206   hzed = thee->pmgp->hzed;
10207
10208   /* Define the total domain size */
10209   xlen = thee->pmgp->xlen;
10210   ylen = thee->pmgp->ylen;
10211   zlen = thee->pmgp->zlen;
10212
10213   /* Define the min/max dimensions */
10214   xmin = thee->pmgp->xcent - (xlen/2.0);

```

```

10215     ymin = thee->pmgp->ycent - (ylen/2.0);
10216     zmin = thee->pmgp->zcent - (zlen/2.0);
10217     xmax = thee->pmgp->xcent + (xlen/2.0);
10218     ymax = thee->pmgp->ycent + (ylen/2.0);
10219     zmax = thee->pmgp->zcent + (zlen/2.0);
10220
10221     /* This is a floating point parameter related to the non-zero nature of the
10222        * bulk ionic strength. If the ionic strength is greater than zero; this
10223        * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
10224        * Otherwise, this parameter is set to 0.0 */
10225     if (ionstr > VPMGSMALL) ionmask = 1.0;
10226     else ionmask = 0.0;
10227
10228     /* Reset the kappa, epsx, epsy, and epsz arrays */
10229     for (i=0; i<(nx*ny*nz); i++) {
10230         thee->kappa[i] = 1.0;
10231         thee->epsx[i] = 1.0;
10232         thee->epsy[i] = 1.0;
10233         thee->epsz[i] = 1.0;
10234     }
10235
10236     /* Loop through the atoms and do assign the dielectric */
10237     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
10238
10239         atom = Valist_getAtom(alist, iatom);
10240         apos = Vatom_getPosition(atom);
10241         arad = Vatom_getRadius(atom);
10242
10243         b = arad - splineWin;
10244         e = arad + splineWin;
10245         e2 = e * e;
10246         e3 = e2 * e;
10247         e4 = e3 * e;
10248         e5 = e4 * e;
10249         b2 = b * b;
10250         b3 = b2 * b;
10251         b4 = b3 * b;
10252         b5 = b4 * b;
10253         denom = pow((e - b), 5.0);
10254         c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10255         c1 = 30.0*e2*b2;
10256         c2 = -30.0*(e2*b + e*b2);
10257         c3 = 10.0*(e2 + 4.0*e*b + b2);
10258         c4 = -15.0*(e + b);
10259         c5 = 6;
10260         c0 = c0/denom;
10261         c1 = c1/denom;
10262         c2 = c2/denom;
10263         c3 = c3/denom;
10264         c4 = c4/denom;
10265         c5 = c5/denom;
10266
10267         b = irad + arad - splineWin;
10268         e = irad + arad + splineWin;
10269         e2 = e * e;
10270         e3 = e2 * e;
10271         e4 = e3 * e;

```

```

10272     e5 = e4 * e;
10273     b2 = b * b;
10274     b3 = b2 * b;
10275     b4 = b3 * b;
10276     b5 = b4 * b;
10277     denom = pow((e - b), 5.0);
10278     ic0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10279     ic1 = 30.0*e2*b2;
10280     ic2 = -30.0*(e2*b + e*b2);
10281     ic3 = 10.0*(e2 + 4.0*e*b + b2);
10282     ic4 = -15.0*(e + b);
10283     ic5 = 6;
10284     ic0 = c0/denom;
10285     ic1 = c1/denom;
10286     ic2 = c2/denom;
10287     ic3 = c3/denom;
10288     ic4 = c4/denom;
10289     ic5 = c5/denom;
10290
10291     /* Make sure we're on the grid */
10292     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
10293         (apos[1]<=ymin) || (apos[1]>=ymax) || \
10294         (apos[2]<=zmin) || (apos[2]>=zmax)) {
10295     if ((thee->pmgp->bclf != BCFL_FOCUS) &&
10296     (thee->pmgp->bclf != BCFL_MAP)) {
10297         Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f,\n
10298 %4.3f) is off the mesh (ignoring):\n",
10299             iatom, apos[0], apos[1], apos[2]);
10300         Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
10301             xmin, xmax);
10302         Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
10303             ymin, ymax);
10304         Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
10305             zmin, zmax);
10306     }
10307     fflush(stderr);
10308
10309 } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
10310
10311     /* Convert the atom position to grid reference frame */
10312     position[0] = apos[0] - xmin;
10313     position[1] = apos[1] - ymin;
10314     position[2] = apos[2] - zmin;
10315
10316     /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
10317      * ASSIGNMENT (Steps #1-3) */
10318     itot = irad + arad + splineWin;
10319     itot2 = VSQR(itot);
10320     ictot = VMAX2(0, (irad + arad - splineWin));
10321     ictot2 = VSQR(ictot);
10322     stot = arad + splineWin;
10323     stot2 = VSQR(stot);
10324     sctot = VMAX2(0, (arad - splineWin));
10325     sctot2 = VSQR(sctot);
10326
10327     /* We'll search over grid points which are in the greater of
10328      * these two radii */

```

```

10329         rtot = VMAX2(itot, stot);
10330         rtot2 = VMAX2(itot2, stot2);
10331         dx = rtot + 0.5*hx;
10332         dy = rtot + 0.5*hy;
10333         dz = rtot + 0.5*hzed;
10334         imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
10335         imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
10336         jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
10337         jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
10338         kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
10339         kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
10340         for (i=imin; i<=imax; i++) {
10341             dx2 = VSQR(position[0] - hx*i);
10342             for (j=jmin; j<=jmax; j++) {
10343                 dy2 = VSQR(position[1] - hy*j);
10344                 for (k=kmin; k<=kmax; k++) {
10345                     dz2 = VSQR(position[2] - k*hzed);
10346
10347             /* ASSIGN CCF */
10348             if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
10349                 dist2 = dz2 + dy2 + dx2;
10350                 if (dist2 >= itot2) {
10351                     ;
10352                 }
10353                 if (dist2 <= ictot2) {
10354                     thee->kappa[IJK(i,j,k)] = 0.0;
10355                 }
10356                 if ((dist2 < itot2) && (dist2 > ictot2)) {
10357                     dist = VSQRT(dist2);
10358                     sm = dist;
10359                     sm2 = dist2;
10360                     sm3 = sm2 * sm;
10361                     sm4 = sm3 * sm;
10362                     sm5 = sm4 * sm;
10363                     value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
10364                         + ic4*sm4 + ic5*sm5;
10365                     if (value > 1.0) {
10366                         value = 1.0;
10367                     } else if (value < 0.0){
10368                         value = 0.0;
10369                     }
10370                     thee->kappa[IJK(i,j,k)] *= value;
10371                 }
10372             }
10373
10374             /* ASSIGN A1CF */
10375             if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
10376                 dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
10377                 if (dist2 >= stot2) {
10378                     thee->epsx[IJK(i,j,k)] *= 1.0;
10379                 }
10380                 if (dist2 <= sctot2) {
10381                     thee->epsx[IJK(i,j,k)] = 0.0;
10382                 }
10383                 if ((dist2 > sctot2) && (dist2 < stot2)) {
10384                     dist = VSQRT(dist2);
10385                     sm = dist;

```

```

10386     sm2 = VSQR(sm);
10387     sm3 = sm2 * sm;
10388     sm4 = sm3 * sm;
10389     sm5 = sm4 * sm;
10390     value = c0 + c1*sm + c2*sm2 + c3*sm3
10391           + c4*sm4 + c5*sm5;
10392     if (value > 1.0) {
10393         value = 1.0;
10394     } else if (value < 0.0){
10395         value = 0.0;
10396     }
10397     thee->epsx[IJK(i,j,k)] *= value;
10398 }
10399 }
10400
10401 /* ASSIGN A2CF */
10402 if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
10403     dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
10404     if (dist2 >= stot2) {
10405         thee->epsy[IJK(i,j,k)] *= 1.0;
10406     }
10407     if (dist2 <= sctot2) {
10408         thee->epsy[IJK(i,j,k)] = 0.0;
10409     }
10410     if ((dist2 > sctot2) && (dist2 < stot2)) {
10411         dist = VSQRT(dist2);
10412         sm = dist;
10413         sm2 = VSQR(sm);
10414         sm3 = sm2 * sm;
10415         sm4 = sm3 * sm;
10416         sm5 = sm4 * sm;
10417         value = c0 + c1*sm + c2*sm2 + c3*sm3
10418           + c4*sm4 + c5*sm5;
10419         if (value > 1.0) {
10420             value = 1.0;
10421         } else if (value < 0.0){
10422             value = 0.0;
10423         }
10424         thee->epsy[IJK(i,j,k)] *= value;
10425     }
10426 }
10427
10428 /* ASSIGN A3CF */
10429 if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
10430     dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
10431     if (dist2 >= stot2) {
10432         thee->epsz[IJK(i,j,k)] *= 1.0;
10433     }
10434     if (dist2 <= sctot2) {
10435         thee->epsz[IJK(i,j,k)] = 0.0;
10436     }
10437     if ((dist2 > sctot2) && (dist2 < stot2)) {
10438         dist = VSQRT(dist2);
10439         sm = dist;
10440         sm2 = dist2;
10441         sm3 = sm2 * sm;
10442         sm4 = sm3 * sm;

```

```

10443                     sm5 = sm4 * sm;
10444                     value = c0 + c1*sm + c2*sm2 + c3*sm3
10445                         + c4*sm4 + c5*sm5;
10446                     if (value > 1.0) {
10447                         value = 1.0;
10448                     } else if (value < 0.0){
10449                         value = 0.0;
10450                     }
10451                     thee->epsz[IJK(i,j,k)] *= value;
10452                 }
10453             }
10454
10455             } /* k loop */
10456         } /* j loop */
10457     } /* i loop */
10458 } /* endif (on the mesh) */
10459 } /* endfor (over all atoms) */
10460
10461 Vnm_print(0, "Vpmgp_fillco: filling coefficient arrays\n");
10462 /* Interpret markings and fill the coefficient arrays */
10463 for (k=0; k<nz; k++) {
10464     for (j=0; j<ny; j++) {
10465         for (i=0; i<nx; i++) {
10466
10467             thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
10468             thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
10469                 + epssp;
10470             thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
10471                 + epssp;
10472             thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
10473                 + epssp;
10474
10475             } /* i loop */
10476         } /* j loop */
10477     } /* k loop */
10478 }
10479
10480 }
10481

```

10.97 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vpmgp.c File Reference

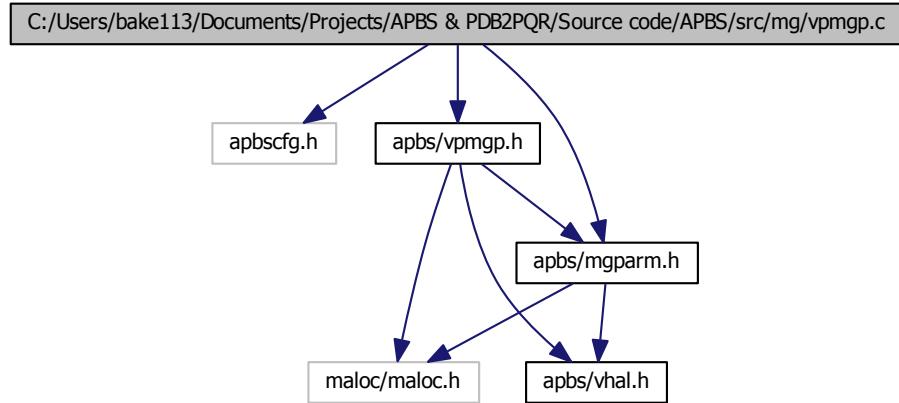
Class Vpmgp methods.

```

#include "apbscfg.h"
#include "apbs/vpmgp.h"
#include "apbs/mgparm.h"

```

Include dependency graph for vpmgp.c:



Functions

- VPUBLIC [Vpmgp * Vpmgp_ctor \(MGparm *mgparm\)](#)
Construct PMG parameter object and initialize to default values.
- VPUBLIC int [Vpmgp_ctor2 \(Vpmgp *thee, MGparm *mgparm\)](#)
FORTRAN stub to construct PMG parameter object and initialize to default values.
- VPUBLIC void [Vpmgp_dtor \(Vpmgp **thee\)](#)
Object destructor.
- VPUBLIC void [Vpmgp_dtor2 \(Vpmgp *thee\)](#)
FORTRAN stub for object destructor.
- VPUBLIC void [Vpmgp_size \(Vpmgp *thee\)](#)
Determine array sizes and parameters for multigrid solver.
- VPRIVATE int **coarsenThis** (nOld)
- VPUBLIC void [Vpmgp_makeCoarse \(int numLevel, int nxOld, int nyOld, int nzOld, int *nxNew, int *nyNew, int *nzNew\)](#)
Coarsen the grid by the desired number of levels and determine the resulting numbers of grid points.

10.97.1 Detailed Description

Class Vpmgp methods.

Author

Nathan Baker

Version

Id:

[vpmgp.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
```

```
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vpmgp.c](#).

10.98 C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/src/mg/vpmgp.c

```
00001
00057 #include "apbscfg.h"
00058 #include "apbs/vpmgp.h"
00059 #include "apbs/mgparm.h"
00060
00061 VEMBED(rcsid="$Id: vpmgp.c 1667 2011-12-02 23:22:02Z pcellis $" )
00062
00063 /* //////////////////////////////// */
00064 // Class Vpmgp: Inlineable methods
00065 #if !defined(VINLINE_VACC)
00066 #endif /* if !defined(VINLINE_VACC) */
00067
00068 /* //////////////////////////////// */
00069 // Class Vpmgp: Non-inlineable methods
00070
00071 /* //////////////////////////////// */
00072 // Routine: Vpmgp_ctor
00073
00074 // Author: Nathan Baker
00075 //
00076 // Author: Nathan Baker
00077 VPUBLIC Vpmgp* Vpmgp_ctor(MGparm *mgparm) {
00078
00079     Vpmgp *thee = VNULL;
00080
00081     /* Set up the structure */
00082     thee = Vmem_malloc(VNULL, 1, sizeof(Vpmgp));
00083     VASSERT( thee != VNULL );
00084     VASSERT( Vpmgp_ctor2(thee, mgparm) );
00085
00086     return thee;
00087 }
00088
00089 /* //////////////////////////////// */
00090 // Routine: Vpmgp_ctor2
00091
00092 //
00093 // Author: Nathan Baker
00094 VPUBLIC int Vpmgp_ctor2(Vpmgp *thee, MGparm *mgparm) {
00095
00096     /* Specified parameters */
00097     thee->nx = mgparm->dime[0];
00098     thee->ny = mgparm->dime[1];
00099     thee->nz = mgparm->dime[2];
00100     thee->hx = mgparm->grid[0];
00101     thee->hy = mgparm->grid[1];
00102     thee->hzed = mgparm->grid[2];
```

```

00104     thee->xlen = ((double) (mgparm->dime[0]-1))*mgparm->grid[0];
00105     thee->ylen = ((double) (mgparm->dime[1]-1))*mgparm->grid[1];
00106     thee->zlen = ((double) (mgparm->dime[2]-1))*mgparm->grid[2];
00107     thee->nlev = mgparm->nlev;
00108
00109     thee->nonlin = mgparm->nonlintype;
00110     thee->meth = mgparm->method;
00111
00112 #ifdef DEBUG_MAC OSX_OCL
00113 #include "mach_chud.h"
00114 if(kOpenCLAvailable)
00115     thee->meth = 4;
00116 #endif
00117
00118     if (thee->nonlin == NONLIN_LPBE) thee->ipkey = IPKEY_LPBE; /* LPBE case */
00119 else if(thee->nonlin == NONLIN_SMPBE) thee->ipkey = IPKEY_SMPBE; /* SMPBE case */
/
00120     else thee->ipkey = IPKEY_NPBE; /* NPBE standard case */
00121
00122     /* Default parameters */
00123     if (mgparm->setetol) { /* If etol is set by the user in APBS input file, then
00124         use this custom-defined etol */
00125         thee->errtol = mgparm->etol;
00126         Vnm_print(1, "   Error tolerance (etol) is now set to user-defined \
00127 value: %g \n", thee->errtol);
00128         Vnm_print(0, "Error tolerance (etol) is now set to user-defined \
00129 value: %g \n", thee->errtol);
00130     } else thee->errtol = 1.0e-6; /* Here are a few comments. Mike had this se
t to
00131     * 1e-9; conventional wisdom sets this at 1e-6 for
00132     * the PBE; Ray Luo sets this at 1e-3 for his
00133     * accelerated PBE (for dynamics, etc.) */
00134     thee->itmax = 200;
00135     thee->istop = 1;
00136     thee->iinfo = 1;           /* I'd recommend either 1 (for debugging LPBE) or 2
00137     (for debugging NPBE), higher values give too much output */
00138
00139     thee->bclf = BCFL_SDH;
00140     thee->key = 0;
00141     thee->iperf = 0;
00142     thee->mgcoar = 2;
00143     thee->mgkey = 0;
00144     thee->nul = 2;
00145     thee->nuz = 2;
00146     thee->mgprol = 0;
00147     thee->mgdisc = 0;
00148     thee->omegal = 19.4e-1;
00149     thee->omegan = 9.0e-1;
00150     thee->ipcon = 3;
00151     thee->irite = 8;
00152     thee->xcent = 0.0;
00153     thee->ycent = 0.0;
00154     thee->zcent = 0.0;
00155
00156     /* Default value for all APBS runs */
00157     thee->mgsmo = 1;
00158     if (thee->nonlin == NONLIN_NPBE || thee->nonlin == NONLIN_SMPBE) {

```

```

00157  /* SMPBE Added - SMPBE needs to mimic NPBE */
00158  Vnm_print(0, "Vpmp_ctor2: Using meth = 1, mgsolv = 0\n");
00159      thee->mgsolv = 0;
00160 } else {
00161 /* Most rigorous (good for testing) */
00162 Vnm_print(0, "Vpmp_ctor2: Using meth = 2, mgsolv = 1\n");
00163     thee->mgsolv = 1;
00164 }
00165
00166 /* TEMPORARY USEAQUA */
00167 /* If we are using aqua, our solution method is either VSOL_CGMGAqua or VSOL_New
tonAqua
00168 * so we need to temporarily override the mgsolve value and set it to 0
00169 */
00170 if(mgparm->useAqua == 1) thee->mgsolv = 0;
00171
00172 return 1;
00173 }
00174
00175 /* /////////////////////////////////
00176 // Routine: Vpmgp_dtor
00177 //
00178 // Author: Nathan Baker
00179 VPUBLIC void Vpmgp_dtor(Vpmgp **thee) {
00180
00181     if ((*thee) != VNULL) {
00182         Vpmgp_dtor2(*thee);
00183         Vmem_free(VNULL, 1, sizeof(Vpmgp), (void **)thee);
00184         (*thee) = VNULL;
00185     }
00186 }
00187
00188 }
00189
00190 /* /////////////////////////////////
00191 // Routine: Vpmgp_dtor2
00192 //
00193 // Author: Nathan Baker
00194 VPUBLIC void Vpmgp_dtor2(Vpmgp *thee) { ; }
00195
00196
00197 VPUBLIC void Vpmgp_size(
00198     Vpmgp *thee
00199 )
00200 {
00201
00202     int num_nf = 0;
00203     int num_narr = 2;
00204     int num_narrc = 27;
00205     int nxf, nyf, nzf, level, num_nf_oper, num_narrc_oper, n_band, nc_band, num_band
, iretot;
00206
00207     thee->nf = thee->nx * thee->ny * thee->nz;
00208     thee->narr = thee->nf;
00209     nxf = thee->nx;
00210     nyf = thee->ny;
00211     nzf = thee->nz;
00212     thee->nxc = thee->nx;
00213

```

```

00214     thee->nyc = thee->ny;
00215     thee->nzc = thee->nz;
00216
00217     for (level=2; level<=thee->nlev; level++) {
00218         Vpmgp_makeCoarse(1, nxf, nyf, nzf, &(thee->nxc), &(thee->nyc), &(thee->nzc)); /
00219         * NAB TO-DO -- implement this function and check which variables need to be passed by reference... */
00220         nxf = thee->nxc;
00221         nyf = thee->nyc;
00222         nzf = thee->nzc;
00223         thee->narr = thee->narr + (nxf * nyf * nzf);
00224     }
00225
00226     thee->nc = thee->nxc * thee->nyc * thee->nzc;
00227     thee->narrc = thee->narr - thee->nf;
00228
00229     /* Box or FEM discretization on fine grid? */
00230     switch (thee->mgdisc) { /* NAB TO-DO: This needs to be changed into an enumeration */
00231         case 0:
00232             num_nf_oper = 4;
00233             break;
00234         case 1:
00235             num_nf_oper = 14;
00236             break;
00237         default:
00238             Vnm_print(2, "Vpmgp_size: Invalid mgdisc value (%d)!\n", thee->mgdisc);
00239             VASSERT(0);
00240     }
00241
00242     /* Galerkin or standard coarsening? */
00243     switch (thee->mgcoar) { /* NAB TO-DO: This needs to be changed into an enumeration */
00244         case 0:
00245             if (thee->mgdisc != 0) {
00246                 Vnm_print(2, "Vpmgp_size: Invalid mgcoar value (%d); must be used with mgdisc 0!\n",
00247                           thee->mgcoar);
00248                 VASSERT(0);
00249             }
00250             num_narrc_oper = 4;
00251             break;
00252         case 1:
00253             if (thee->mgdisc != 0) {
00254                 Vnm_print(2, "Vpmgp_size: Invalid mgcoar value (%d); must be used with mgdisc 0!\n",
00255                           thee->mgcoar);
00256                 VASSERT(0);
00257             }
00258             num_narrc_oper = 14;
00259             break;
00260         default:
00261             Vnm_print(2, "Vpmgp_size: Invalid mgcoar value (%d)!\n", thee->mgcoar);
00262             VASSERT(0);
00263     }
00264

```

```

00265 /* LINPACK storage on coarse grid */
00266 switch (thee->mgsolv) { /* NAB TO-DO: This needs to be changed into an enumeration */
00267     case 0:
00268         n_band = 0;
00269         break;
00270     case 1:
00271         if ( ( (thee->mgcoar == 0) || (thee->mgcoar == 1) ) && (thee->mgdisc == 0) ) {
00272             num_band = 1 + (thee->nxc-2)*(thee->nyc-2);
00273         } else {
00274             num_band = 1 + (thee->nxc-2)*(thee->nyc-2) + (thee->nxc-2) + 1;
00275         }
00276         nc_band = (thee->nxc-2)*(thee->nyc-2)*(thee->nzc-2);
00277         n_band = nc_band * num_band;
00278         break;
00279     default:
00280         Vnm_print(2, "Vpmgp_size: Invalid mgsolv value (%d)!\n", thee->mgsolv);
00281         VASSERT(0);
00282     }
00283
00284     /* Real storage parameters */
00285     thee->n_rpc = 100*(thee->nlev+1);
00286
00287     /* Resulting total required for real storage */
00288     thee->nwk = num_narr*thee->narr + (num_nf + num_nf_oper)*thee->nf + (num_narrc
00289     + num_narrc_oper)*thee->narrc + n_band + thee->n_rpc;
00290
00291     /* Integer storage parameters */
00292     thee->n_iz = 50*(thee->nlev+1);
00293     thee->n_ipc = 100*(thee->nlev+1);
00294     thee->niwk = thee->n_iz + thee->n_ipc;
00295 }
00296 VPRIPRIVATE int coarsenThis(nOld) {
00297
00298     int nOut;
00299
00300     nOut = (nOld - 1) / 2 + 1;
00301
00302     if (((nOut-1)*2) != (nOld-1)) {
00303         Vnm_print(2, "Vpmgp_makeCoarse: Warning! The grid dimensions you have chosen
00304         are not consistent with the nlev you have specified!\n");
00305         Vnm_print(2, "Vpmgp_makeCoarse: This calculation will only work if you are run
00306         ning with mg-dummy type.\n");
00307     }
00308     if (nOut < 1) {
00309         Vnm_print(2, "D'oh! You coarsened the grid below zero! How did you do that?\n
00310     ");
00311     VASSERT(0);
00312 }
00313
00314 VPUBLIC void Vpmgp_makeCoarse(
00315     int numLevel,
00316     int nxOld,

```

```
00317 int nyOld,
00318 int nzOld,
00319 int *nxNew,
00320 int *nyNew,
00321 int *nzNew
00322 )
00323 {
00324 int nxtmp, nyttmp, nztmp, iLevel;
00325
00326 for (iLevel=0; iLevel<numLevel; iLevel++) {
00327     nxtmp = *nxNew;
00328     nyttmp = *nyNew;
00329     nztmp = *nzNew;
00330     *nxNew = coarsenThis(nxtmp);
00331     *nyNew = coarsenThis(nyttmp);
00332     *nzNew = coarsenThis(nztmp);
00333 }
00334
00335
00336 }
```